Quantum Circuit Design using Genetic Algorithm for Melody Generation with Quantum Computing

Tatsunori Hirai

Komazawa University
thirai@komazawa-u.ac.jp

Abstract. In this paper, we explore the potential of quantum computing for music generation, particularly for generating melodies. We propose a method of designing quantum circuits with genetic algorithms for melody generation. Our method allows for the generation of subsequent musical notes for arbitrary input notes and the production of melodies of varying lengths based on the transition distribution between melodies in the training data. We compared the accuracy of a quantum computer in predicting the subsequent note based on the training data with that of a classical computer. Our results demonstrate the potential of quantum computing for melody generation.

Keywords: Melody generation; quantum computing; genetic algorithm

1 Introduction

The first instance of music being automatically generated by a computer was the "ILLIAC Suite, for String Quartet," composed by ILLIAC I in 1957 [1]. Since then, researchers have been investigating music generation techniques, including automatic composition, from the early days of computing to the present day

The realization of quantum computing, expected to be the next-generation computing paradigm, is becoming increasingly feasible. As of May 2023, Noisy Intermediate-Scale Quantum Computers (NISQ), a quantum computer designed with the assumption of the inclusion of various types of noise, have been realized with hundreds of qubits and are available on the cloud.

To discuss the need for a quantum computer, it is important to explore its potential applications. The extent and fields in which quantum computers will be useful remain uncertain at present. It is also unclear whether quantum computers can be considered superior to classical computers.

In this paper, we investigate the potential applications of quantum computing for the task of music generation, a domain that has been extensively studied using classical computing methods. We specifically assess the feasibility of music generation using current gated quantum computer architectures and propose a novel approach for designing quantum circuits by employing genetic algorithms.

2 Related Work

The utilization of quantum computers for musical expression, termed "quantum music," has been the subject of multiple investigations in recent years, reflecting a growing interest in this interdisciplinary field. For example, Kirke et al. proposed Q-MUSE, a live performance music system where output sound is modulated by altering input parameters of a quantum computer through a button controller and gesture controller [2]. Additionally, Clemente et al. introduced a keyboard, termed "qeyboard," in which sound parameters are controlled by quantum circuits [3].

The QuTune project [4] is a research project focused on generating music through quantum computing, culminating in the organization of the first international symposium on quantum music in 2021. The QuTune team has produced several technical papers and comprehensive reviews on this subject. In previously published review articles [5], [6], the authors discuss the fundamentals of quantum computers and computer music, introducing specific applications such as the Quantum Vocal Synthesizer and the Quantum Walk Sequencer. Notably, the Quantum Walk Sequencer is a sequencer that facilitates note-to-note transitions using a quantum random walk [7]. This approach, which utilizes quantum circuits to represent note transitions, offers valuable insights for melody generation. Furthermore, the QuTune team is developing a music generation system that incorporates a quantum natural language processing (QNLP) approach, integrating quantum computing within a natural language processing framework [8].

Kirke proposed the hybrid music generation system qGEN, which integrates a gated quantum computer and a quantum annealing machine [9]. qGEN produces music by combining GATEMEL, a melody generator utilizing a gated quantum computer, with qHARMONY, a system that generates accompaniments for given melodies using a quantum annealing machine. Souma proposed quantum live coding, a method for generating improvised music based on gated quantum algorithms [10]. This approach involves producing melodies by connecting consecutive notes through quantum entanglement.

The recent efforts applying quantum computing to musical expression outlined in this section signify the emerging development of this research domain.

3 Possibilities of Quantum Computing in Music Generation

A key feature of quantum computers is the superposition state of qubits. Upon measurement, the superposition collapses, yielding a 0 or 1 state similar to classical bits. Quantum algorithms leverage superposition states, representing all possible inputs, to increase the likelihood of obtaining the desired outcome through measurement.

In this context, we explore the application of quantum computers to the task of music generation. It is essential to recognize that in music, there is no definitive "correct" answer, and pursuing a singular answer might not be ideal, especially in the context of music generation. The pursuit of a single correct answer in generative tasks is unlikely to be accomplished, regardless of the efficiency of search algorithms developed. In music, there are many sequences and combinations of sounds that are considered undesirable by many people. Therefore, it is desirable to develop an algorithm that can avoid such results when generating music.

Table 1. A binary representation of a note name.

note name	C	D	E	F	G	A	B	R
binary digits	000	001	010	011	100	101	110	111

The superposition state of qubits in a quantum computer enables the representation of all possible melody combinations simultaneously when generating melodies. As there are numerous undesirable note combinations included in all possible melodies, designing the quantum algorithm such that the probability of measuring such an undesired combination is reduced is a potential approach. However, the final measurable result is just one of all possible combinations, meaning that even though the quantum computer considers all combinations simultaneously, the resulting melody is only one.

Due to the distinct features of quantum and classical computers, it may be possible to achieve efficient results by incorporating a quantum computer, depending on the algorithm being used. An example of a successful application of quantum computing is the generation of random numbers. Quantum computing can successfully generate true random numbers, thanks to the probabilistic nature of qubits. However, since precision is not crucial in music generation, the significance of introducing a quantum computer remains debatable. In this paper, we propose a melody generation algorithm that utilizes quantum circuits to replicate the note transitions observed in training data, taking advantage of the characteristic of true random number generation in quantum computers.

4 Data Representation for Quantum Circuit Design

In this chapter, we introduce the data representation for melody generation with a quantum computer. Generating melodies using quantum circuits necessitates determining an appropriate method to represent notes and quantum gates numerically.

4.1 Data Representation of Note Names

Here, we have simplified the problem to its core elements to enable clear observation of the behavior of the quantum circuit. The problem is set up by considering only the essential elements that compose a melody, namely note names. To handle melodies as simply as possible, we represent notes using a 3-bit binary number that only represents the note name. Specifically, the note names are represented as a 3-bit binary number, with C being represented as 000. A total of 8 note names are used, including seven types of notes from C (000) to B (110) and a rest represented as R (111). Note durations are not considered, and all notes are assumed to have a fixed duration of one quarter note. The binary representation and corresponding note names are presented in Table 1. When applying this data representation to a quantum computer, the problem is set up such that a 3-qubit quantum circuit generates the next note based on the input note.

4.2 Data Representation of Quantum Gates

There are various types of quantum gates that compose quantum circuits. To realize melody generation in the simplest problem setting, we selected 4 basic types of gates

Table 2. Numerical representation of quantum gates.

quantum gate	numerical representation
H gate (register 0)	0
H gate (register 1)	1
H gate (register 2)	2
X-axis rotation gate : $\pi/4$ (register 0)	3
X-axis rotation gate : $\pi/4$ (register 1)	4
X-axis rotation gate : $\pi/4$ (register 2)	5
CX gate (register 0 to 1)	6
CX gate (register 0 to 2)	7
CX gate (register 1 to 2)	8
CX gate (register 1 to 0)	9
CX gate (register 2 to 0)	10 (A)
CX gate (register 2 to 1)	11 (B)
CCX gate (register 0,1 to 2)	12 (C)
CCX gate (register 0,2 to 1)	13 (D)
CCX gate (register 2,1 to 0)	14 (E)
no gate	15 (F)

and numerically represented 16 different gate placement patterns. These patterns include variations in which qubits the gates act upon among the 3 input registers. The 16 possible gate arrangements include a state with no gates, and each arrangement is represented by a unique hexadecimal number. For each quantum circuit representation, we use 8-digit hexadecimal numbers to represent the gate arrangement. This enables representation of quantum circuits with 0-8 gate combinations. Table 2 shows the numerical representation of quantum gates and their corresponding gate arrangement patterns.

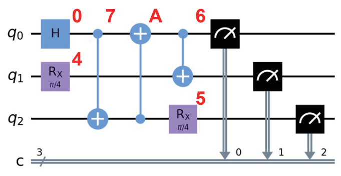

Fig. 1. Example of quantum circuit represented by “04F7A6F5.”

Using the hexadecimal numerical representation of quantum gates in Table 2, a quantum circuit can be represented by an 8-digit hexadecimal number. For example, the circuit diagram represented by “04F7A6F5” is shown in Fig.1. According to the correspondence shown in Table 2, F represents no gates, so in this case, the quantum circuit consists of 6 gates. The 0 represents the H gate (Hadamard gate) applied to register 0 (the register corresponding to q_0 in Fig.1), which puts the qubit in a superposition. The 4 and 5 are rotation gates around the X-axis applied to registers 1 and 2, respectively, with rotation angles of $\pi/4$. The 7, A, and 6 are all CX gates (controlled-NOT gates), which enable interactions between the registers.

Table 3. Measurement results (for 200 shots) when $|000\rangle$ (C) is input to the quantum circuit shown in Fig.1.

output	000(C)	001(D)	010(E)	011(F)	100(G)	101(A)	110(B)	111(R)
measurement count	80	0	17	0	93	0	10	0

There exist many more varieties of quantum gates beyond those listed here. While more complex quantum circuits can be expressed in the same numerical framework by assigning numbers to other gates, this paper prioritizes simplicity, and only the basic gates listed here will be utilized. Experiments with more complex quantum circuit configurations are also possible, but are left as future work, as current quantum computers are highly susceptible to errors in constructing such circuits.

To ensure the usefulness of quantum circuits, it is necessary to take advantage of the superposition of quantum states. If superposition is not utilized, the quantum circuit operation can be reproduced using a classical computer, making the use of quantum circuits meaningless. Hence, we operate the H gate once for all inputs of registers 0 to 2, and then add other gates represented by 8-digit hexadecimal numbers to utilize superposition.

4.3 Generation of subsequent Note with Quantum Circuit

The problem of generating a subsequent note for an input note can be represented by the input/output of data to/from a quantum circuit, achieved by combining the data representation of note names and quantum gate representation. To generate a melody, the initial note is determined and input to the quantum circuit to generate subsequent notes by measuring the output qubits. The process is repeated by inputting the generated subsequent note as input data to the quantum circuit to generate further notes, thus completing the melody.

Table 3 shows the measurement results (for 200 shots) obtained when the input note C, represented as $|000\rangle$, is used as the input to the quantum circuit shown in Fig.1. If this quantum circuit were used to generate the subsequent note, it would only output notes corresponding to the states of C, E, G, or B. The results presented in Table 3 are obtained when $|000\rangle$, corresponding to C, is directly input to the quantum circuit in Fig.1 without using the H gates at the beginning.

We attempted two methods for generating quantum circuits:

- **Training-A**: train one circuit for each type of input note.
- **Training-B**: train a single circuit for all input-output combinations.

In a case circuit is trained for each input note (Training-A), the C circuit generates the subsequent note from the input C, and the D circuit generates the subsequent note from the input D. The design process of each quantum circuit will be presented in the next chapter.

5 Designing Quantum Circuits with Genetic Algorithms

Designing a quantum circuit is equivalent to determining a quantum algorithm. It affects the quality of the generation results. There are several possible approaches to design quantum circuits, and one example is to use a H gate in every register to represent a superposition of all combinations, resulting in a circuit with completely random outputs.

Manual gate determination in quantum circuits can lead to inflexible designs with fixed outcome patterns. Thus, this paper investigates a flexible circuit design method using training data. We prepare arbitrary melodies as training data and search for optimal quantum gate combinations that reproduce the desired output distribution based on that training data. This approach establishes a well-defined criterion for designing quantum circuits that accurately emulate the input-output relationship in the training data.

We utilize a genetic algorithm to explore quantum gate combinations, aiming to minimize the discrepancy between the quantum circuit's output and the training data's note transition distribution. The genetic algorithm runs on a classical computer, while the quantum computer generates subsequent notes based on input notes, making our approach a hybrid method. Moreover, training processes are conducted using a quantum circuit simulator on a classical computer, while the actual quantum computer is used for generating melodies with the resulting quantum circuit. Using a quantum computer during circuit design is feasible, but the extensive trials needed for training make it challenging within a realistic time frame, given current capabilities. Future advancements in quantum computing may address these challenges.

5.1 Preparation of Training Data

The training data is created from the note sequences present in pre-existing musical compositions. The types of notes that can be handled by the algorithm proposed in this study are limited to eight types, from C to B and R. Therefore, the melody used for training is composed solely of simple quarter notes in the key of C major. In this study, melodies from three pieces, namely "Twinkle, Twinkle, Little Star," "Tulip," and "Froggy's Song," were chosen for the purpose of training.

For instance, when creating training data based on the initial melody of "Twinkle, Twinkle, Little Star," the melody can be represented as "C→C→G→G→A→A→G→R." Consequently, the note following C is exclusively either C or G, with no transition to other notes. To replicate this pattern, an ideal quantum circuit would measure C (000) and G (100) with a 50% probability each for an input of C (000). This input-output relationship can be achieved using a quantum circuit with a single H gate in the second register. For the actual training process with a more intricate output distribution, we automate quantum circuit design using a genetic algorithm instead of manual configuration.

The training data in this study is composed of transition probabilities between note names. As a result, the generated quantum circuit serves as a model for generating subsequent notes utilizing a bi-gram approach. The actual training data consists of a single matrix representing the transition probabilities between note names found in the three selected pieces. The transition probabilities for the note names utilized as training data are presented in Table 4. According to these transition probabilities, when note A is input, the likelihood of G being generated as the subsequent note is the highest at 0.57, followed by A at 0.43, while the probabilities for the other notes are 0. Notably, none of the melodies in the training data included the note B.

The training data can be readily expanded by incorporating a greater number of musical pieces. However, the melody generation approach presented in this paper is limited

Table 4. Training data (transition probability of note names).

		subsequent note							
		C	D	E	F	G	A	B	R
input note	C	0.10	0.38	0	0	0.10	0	0	0.43
	D	0.33	0.14	0.38	0	0	0	0	0.14
	E	0.04	0.40	0.20	0.12	0.04	0	0	0.20
	F	0	0	0.58	0.33	0.08	0	0	0
	G	0	0	0.17	0.17	0.28	0.22	0	0.17
	A	0	0	0	0	0.57	0.43	0	0
	B	0	0	0	0	0	0	0	0
	R	0.50	0	0.11	0.11	0.28	0	0	0

to eight note types. Given that all notes are quarter notes, the range of applicable pieces is somewhat restricted. While it is feasible to learn more complex melody distributions within the same framework by eliminating constraints on note value and increasing the diversity of manageable note names, such considerations are beyond the scope of this paper. In order to validate the efficacy of automating quantum circuit design, it is crucial to initially establish a simplified problem framework to the greatest extent possible.

5.2 Details of Genetic Algorithm

The genetic algorithm is utilized to learn a sequence of 8-digit hexadecimal numbers representing quantum gate combinations, as introduced in Section 4.2. A randomly initialized sequence of 8-digit hexadecimal numbers is treated as an individual within the genetic algorithm, with each digit corresponding to a gene. The process was repeated for 100 generations, with 1000 individuals in each generation subjected to tournament selection, two-point crossover, and mutation steps, ensuring a preference for individuals exhibiting high fitness. The fitness value is computed by taking the mean squared error between the output distribution of 200 shots from a quantum circuit, as shown in Table 3, and the target output note distribution (Table 4) used for training. The crossover probability was set to 0.5, the probability of individual mutation was set to 0.2, and the gene mutation probability was set to 0.05.

In the case that no gates are present to compose a quantum circuit, the gene sequence is represented as "FFFFFFFF." In this state, each H gate operates once on every input qubit, resulting in a superposition of all possible states, which implies that the output may consist of any of the eight notes. Building upon this initial state, the quantum circuit's gate configuration is trained by altering the gene sequence and incorporating quantum gates corresponding to the sequence modifications, thereby generating outputs that more closely align with the training data. The more closely the distribution of outputs aligns with the training data, the more desirable the design of the quantum circuit can become to achieve the desired output distribution.

One of a quantum circuit designed by the genetic algorithm utilizing the training data (Table 4) is depicted in Fig.2. Given that the quantum circuit in Fig.2 is designed for the input note R (rest), X gates are placed at all the registers before the remaining gates to set the input bit value to 1. Consequently, the input state $|000\rangle$ transitions to

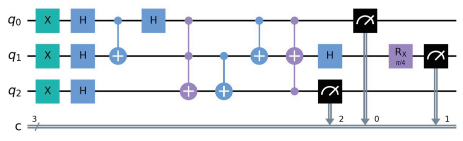

Fig. 2. A quantum circuit designed by genetic algorithm ("60C86D14"), with the input note R (Training-A).

$|111\rangle$, followed by the H gates operating on all qubits and the subsequent operations, as represented by the 8-digit hexadecimal numbers. For Training-B, where a common quantum circuit is used for all note inputs, the configuration of the initial X gates varies for each input note, while the circuit represented by the 8-digit hexadecimal number remains consistent.

Although the quantum circuit in Fig.2 contains redundant gate arrangements, such as consecutive utilization of CX and CCX gates, its output distribution closely approximates the training data $[0.5, 0, 0.11, 0.11, 0.28, 0, 0, 0]$ (bottom row of Table 4). The output result will be described in Section 5.3.

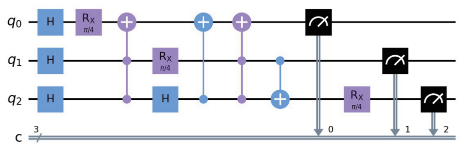

Fig. 3. A quantum circuit designed by genetic algorithm ("3D24ADB5"), for all input/output note combinations (Training-B). X gates for distinguishing the types of input notes are omitted here.

In the Training-B setting, an integrated version of a quantum circuit capable of handling all input notes within a single circuit was trained. The training result for all input notes in a single quantum circuit are presented in Fig.3. This quantum circuit accommodates all input notes by inserting X gates that correspond to each input note name at the beginning of the quantum circuit. For input notes other than C, X gates are applied to modify the input qubits to the corresponding input state.

5.3 Generation of subsequent Note by Trained Quantum Circuits

A subsequent note corresponding to the input note was generated using a quantum circuit that had been trained on the given data through a genetic algorithm. First, as a result of training distinct quantum circuits for each input note (Training-A), Table 5 displays the outputs obtained by executing 100 shots for each of the eight different circuits corresponding to each input note respectively. It is important to note that, during the actual generation step, only one shot is executed, and the measured output serves as the generated subsequent note. Table 5 displays the distribution of outputs obtained by

Table 5. The results of generating subsequent notes with the Training-A setting, using quantum circuits trained for each input note (100 shots). Each row corresponds to a trained quantum circuit.

		Generated subsequent notes (measurement counts)							
		C	D	E	F	G	A	B	R
quantum circuit	C	29	25	0	0	25	0	0	21
	D	27	22	28	2	0	6	0	15
	E	4	30	15	4	12	10	2	23
	F	0	0	55	45	0	0	0	0
	G	6	1	26	20	19	23	3	2
	A	0	0	0	0	49	51	0	0
	B	8	11	12	16	6	18	17	12
	R	53	0	4	0	36	0	7	0

Table 6. The results of generating subsequent notes with the Training-B setting, using a single quantum circuit (Fig.3) trained for all input notes (100 shots).

		Generated subsequent notes (measurement counts)							
		C	D	E	F	G	A	B	R
input note	C	28	21	3	3	2	3	24	16
	D	24	35	15	11	0	0	5	10
	E	21	22	4	3	5	9	16	20
	F	14	15	26	22	9	14	0	0
	G	2	4	25	21	20	19	5	4
	A	0	0	15	10	26	28	10	11
	B	4	2	16	25	25	21	3	4
	R	7	9	0	0	19	12	29	24

executing 100 shots. Although the results vary with each execution, the distribution of subsequent notes closely resembles that presented in the training data (Table 4).

Subsequently, the output obtained by executing 100 shots for each of the eight distinct input notes on a single quantum circuit trained for all notes (Training-B) is presented in Table 6. In this case, the quantum circuits, represented as 8-digit hexadecimal numbers (i.e., “3D24ASB5”), remain the same for all inputs. As a result, the subtle bias in the distribution for each input note could not be trained as effectively as when employing different circuits for each input note.

A comparison of Table 4 and 5, or Table 4 and 6, reveals the extent to which the output distribution of the quantum circuits resembles the training data distribution. In the training data, the transition probability to note B was zero for all input notes; however, the resulting quantum circuits did allow for transitions to note B. This was particularly noticeable when training a single quantum circuit for all input notes (Training-B). For the input note B, all outputs were measured in both the results of Table 5 and 6, since the distribution of outputs to be trained comprised only zeros. Therefore, this result is not an erroneous.

It should be noted that exact replication of the output distribution is impossible for a quantum computer, as the outcome varies with each execution.

Table 7. Comparison of error rates in reproducing note transitions. Comparison of random number generation with classical computers, quantum circuit for each input (Training-A), and quantum circuit for all input (Training-B).

Random number generated with classical computer	Quantum circuit for each input note (Trainig-A)	Quantum circuit for all input notes (Trainig-B)
6.90×10^{-4}	4.63×10^{-3}	1.91×10^{-2}

5.4 Comparison of Note Transition Reproduction

In this section, we investigate the extent to which the generation of subsequent notes by quantum circuits can accurately reproduce the note transitions in the training data. To make a comparison, we also include results obtained from a classical computer that generates subsequent notes according to the distribution of training data using random numbers. For generating random numbers with a classical computer, we utilized the random module in the Python standard library.

The error rate R of reproducing subsequent note is defined as follows:

$$R = \sqrt{\frac{1}{64}\sum_{i=1}^{8}\sum_{j=1}^{8}(t_{ij} - x_{ij})^2} \tag{1}$$

where t_{ij} denotes the note transition probability of the training data from note i to note j, and x_{ij} denotes the probability of transition with each method to be compared. The x_{ij} is calculated based on the distribution obtained from 100 generated subsequent notes for each input note (i.e. Table 5 and Table 6). The error rate of subsequent note reproduction corresponds to the mean square error (MSE).

We generated x_{ij} for the classical computer by generating 100 random numbers for each input note. The subsequent notes are then determined by comparing the generated random numbers with the distribution of the training data. In other words, subsequent notes are directly generated from the note transition probabilities of the training data.

Table 7 shows a comparison of error rate R in reproducing subsequent notes using each circuit/computer, where smaller R values indicate higher reproducibility. The classical computer achieved high accuracy by directly using the training data's transition probability distribution. On the other hand, in the case of quantum circuits, the error rate was lower when using different circuits for each input note (Training-A setting), suggesting that varying circuits yield better output distribution. Due to the randomness of all methods, the error rate of reproduction varies to some extent with each execution.

Note that the training data did not include note B, so the values for the note transitions from note B were excluded from the calculation in Table 7 (x_{7j} and x_{i7} were set to 0). This is because including note B would simply increase the error rate for all methods, thus it was excluded from the analysis.

This comparison was verified using the Qasm simulator, a quantum circuit simulator that uses a classical computer. It should be noted that in the case of an actual quantum computer, errors in the quantum circuit must also be considered.

6 Melody Generation Results

We utilized the quantum circuit designed using the above methodology to generate musical melodies. The process starts with selecting the first note and obtaining the output from the corresponding quantum circuit specifically designed for the input note. The next note in the sequence is determined by inputting the initial output note into the quantum circuit that corresponds to it, and this process is repeated iteratively. The choice of the first note and the number of iterations can be arbitrarily determined, contingent on the desired length of the melody.

Fig.4 shows scores of melodies generated by the quantum computer. Two 4-measure melodies were generated for each quantum circuit design pattern (Training-A and Training-B), starting with C and G. We employed the ibm_bogota quantum computer, provided by IBM Q, which can operate up to five qubits. It should be noted that when using an actual quantum computer, errors may occasionally arise, leading to note transitions that exhibit zero probabilities in Table 5 (C→A and F→C in Fig.4 bottom-left) and Table 6 (D→G in Fig.4 top-right).

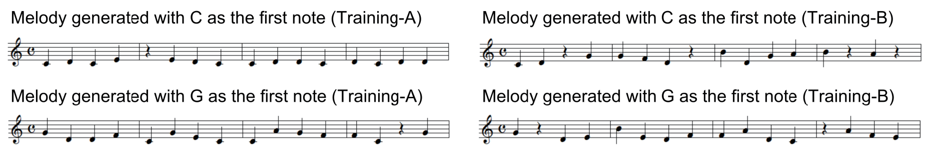
Melody generated with C as the first note (Training-A)
Melody generated with C as the first note (Training-B)
Melody generated with G as the first note (Training-A)
Melody generated with G as the first note (Training-B)

Fig. 4. Examples of melody generation utilizing an actual quantum computer. Left: results employing eight different trained quantum circuits for each input note (Training-A). Right: results employing single trained quantum circuit for all input note names (Training-B).

The execution time required for generating a melody utilizing an actual quantum computer is dependent on the waiting time experienced by the quantum computer when performing the task at a particular instance. In our experiments, generating a 16-note melody necessitated approximately 15 to 30 minutes. This observation does not necessarily suggest that the process inherently requires an extended execution time; instead, it reflects the current limitations of quantum computing resources accessible through cloud services, which result in the increased execution time.

7 Conclusions

In this paper, we explored the potential application of quantum computers in music generation and proposed a genetic algorithm-based method for designing quantum circuits to generate melodies. We compared the accuracy of reproducing subsequent notes between quantum and classical computers. The results of new melodies generated by quantum circuits trained on specific musical pieces are presented. Although a similar melody generation can be achieved with a classical computer, the distinct difference lies in the randomness of the outcome. Nevertheless, we demonstrated that the process of generating subsequent notes, as performed by classical computers, can also be implemented with a quantum computer without manual design of quantum circuits. The input/output relationship of notes is successfully represented using quantum circuits, indicating that music generation algorithms on classical computers may also be feasible in quantum circuits. We aim to further investigate the possibilities of quantum computing in the domain of music generation in future research.

In order to maintain a simple problem setup, the melody generation approach proposed in this study employed a limited number of available note types and lengths, as well as restricted types and quantities of quantum gates. Although these constraints can be easily mitigated, the primary focus of this paper was not to increase the complexity of the results. Instead, the central objective of this work was to explore the potential of quantum computing for music generation.

Quantum computing is currently in the early stages of development. Future quantum programming paradigms may not rely on quantum circuits. Moreover, the optimal design of applications may experience considerable transformations in response to changes in the utilization of quantum computing. We will persist in investigating the potential application of quantum computers for music generation. In our future research, we aim to utilize quantum computing to enhance the expressiveness of music generation in ways that have not been achievable with classical computers.

References

1. Hiller, L. Isaacson, L.M.: Illiac suite, for string quartet. Vol.30, No.3, New Music Edition (1957)
2. Kirke, A., Shadbolt, P., Neville, A., Antoine, A., Miranda, E.R.: Q-Muse: A quantum computer music system designed for a performance for orchestra, electronics and live internet-connected photonic quantum computer In: Proceedings of the 9th Conference on Interdisciplinary Musicology (2014)
3. Clemente, G., Crippa, A., Jansen, K., Tüysüz, C.: New Directions in Quantum Music: Concepts for a Quantum Keyboard and the Sound of the Ising Model. Quantum Computer Music: Foundations, Methods and Advanced Concepts, Cham: Springer International Publishing, pp.433–445 (2022)
4. Eduardo R. M. Bob C., et al.: QuTune Project. https://iccmr-quantum.github.io/.
5. Miranda, E.R.: Quantum Computer: Hello, Music!. Handbook of Artificial Intelligence for Music, pp.963–994 (2021)
6. Miranda, E.R. Bask, S.T.: Quantum Computer Music: Foundations and Initial Experiments. Quantum Computer Music: Foundations, Methods and Advanced Concepts, Cham: Springer International Publishing, pp.43–67 (2022)
7. Aharonov, Y., Davidovich, L., Zagury, N.: Quantum random walks. Physical Review A, Vol.48, No.2, pp.1687–1690 (1993)
8. Miranda, E.R. Yeung, R. Pearson, A. Meichanetzidis, K. Coecke, B.: A Quantum Natural Language Processing Approach to Musical Intelligence. Quantum Computer Music: Foundations, Methods and Advanced Concepts, Cham: Springer International Publishing, pp.313–356 (2022)
9. Kirke, A.: Programming gate-based hardware quantum computers for music. Physical Review A, Vol.48, No.2, pp.1687–1690 (1993)
10. Souma S.: Exploring the Application of Gate-type Quantum Computational Algorithm for Music Creation and Performance. Quantum Computer Music: Foundations, Methods and Advanced Concepts, Cham: Springer International Publishing, pp.88–103 (2022)