

Melody Generation with Quantum Computing

—Automatic Quantum Circuit Design Using a Genetic Algorithm—

Tatsunori Hirai

Abstract

In this paper, we present melody generation method using quantum computing. The topic of music generation has been of interest in computer science since the early days of computing. With the advent of quantum computing, this paper explores the potential of using a quantum computer for music generation, particularly for generating melodies. In this research, we propose a method of designing quantum circuits with genetic algorithms for melody generation. Our proposed 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 *.

Key words: Melody generation; quantum computing; genetic algorithm

Introduction

Music generated automatically by a computer first appeared in the form of the “ILLIAC Suite, for String Quartet,” composed by ILLIAC I in 1957¹. 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. IBM is currently developing IBM Q, a cloud-based quantum computer, and has announced plans to build a quantum computer with over 4000 qubits by 2025².

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

Quantum computers have demonstrated significant potential in various applications, such as quantum chemical calculations, combinatorial optimization problems, financial simulations, and quantum machine learning. However, it is important to note that numerous research problems within the computer science field may not necessitate the implementation of quantum computing. In many cases, these tasks can be

* This paper is an extended version of T. Hirai: "Quantum Circuit Design using Genetic Algorithm for Melody Generation with Quantum Computing," in the proceedings of the 16th International Symposium on Computer Music Multidisciplinary Research (CMMR 2023).

1 Hiller, L. Isaacson, L.M.: Illiac suite, for string quartet. Vol.30, No.3, New Music Edition (1957).

2 Gambetta J.: Expanding the IBM Quantum roadmap to anticipate the future of quantum-centric supercomputing. <https://research.ibm.com/blog/ibm-quantum-roadmap-2025>.

accomplished more accurately and efficiently using classical computing methods alone.

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.

Qubits and Gated Quantum Computers

A quantum computer is a computing device that employs quantum bits, or qubits, which can represent superpositions of quantum states. In the context of quantum computing, bits utilized by classical computers are referred to as classical bits, with each bit assuming a state of either 0 or 1. Conversely, a qubit can represent a superposition of both 0 and 1 states. For instance, two qubits can simultaneously exhibit four superposed states: “00,” “01,” “10,” and “11.”

Leveraging the unique property of qubits, which enables the simultaneous processing of multiple input combinations, quantum algorithms can efficiently obtain desired outcomes. However, qubits exhibit a characteristic wherein their state collapses to either a 0 or 1 upon measurement. Consequently, designing a quantum circuit or algorithm that consistently yields the desired result proves challenging, and in practice, the outcome is determined probabilistically by analyzing output biases across multiple executions.

Current programming methodologies for quantum computers predominantly focus on gated quantum computing, which involves constructing quantum circuits through the combination of quantum gates that execute diverse operations on qubits. A prime example of current quantum programming is the use of Qiskit, a framework provided by IBM that operates on IBM Q. Qiskit is a Python-based framework, with the program being described in Python. However, the commands within Qiskit pertain to the assembly of quantum circuits that amalgamate quantum gates and simulate quantum computation on these constructs. Hence, developing a program that accomplishes the desired task using Qiskit necessitates not only programming expertise but also a comprehensive understanding of quantum circuits and quantum computation, making it a challenging endeavor.

Designing quantum circuits using Qiskit presents a significant challenge in current quantum programming, as it demands an intricate combination of operations on superposed states. An additional factor contributing to this complexity is the stochastic nature of quantum superposition states. In classical bits, under the assumption of no error effects, a state of 0 remains 0 and a state of 1 remains 1. Additionally, classical bits are designed with error-correcting mechanisms to ensure the integrity of the information. Conversely, when processing superposed qubit states using the same quantum circuit, the superposition state collapses upon measurement, resulting in either 0 or 1 being measured probabilistically. Consequently, execution results can differ even when employing the same quantum circuit. Moreover, with current NISQ, an increase in gate complexity within a quantum circuit leads to higher error rates and a greater likelihood of obtaining inaccurate computation results. This complex relationship between cause and effect also poses a significant obstacle to the practical implementation of quantum computers.

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 ³. Additionally, Clemente et al. introduced a keyboard, termed “qeyboard,” in which sound parameters are controlled by quantum circuits ⁴.

Weimer’s “Listen to Quantum Computer Music” ⁵ explored generating music by translating the quantum gates comprising well-known quantum algorithms, such as Grover’s algorithm ⁶ and Shor’s algorithm ⁷, into musical notes. Quantum circuit diagrams are often likened to musical scores due to their left-to-right arrangement of gates. Weimer’s approach involved interpreting the quantum circuit as a musical score and generating music accordingly. Weimer actually generated music by interpreting the quantum circuit as a musical score. This method diverges from quantum music, as it does not directly generate music through quantum algorithms but rather converts quantum circuits themselves into musical notation.

The QuTune project ⁸ is a research project focused on generating music through quantum computing, culminating in the organization of the first international symposium on quantum music, the ISQCMC, the first international symposium on quantum computing and musical creativity held online in November 2021. The QuTune team has produced several technical papers and comprehensive reviews on this subject. In previously published review articles ⁹, ¹⁰, 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, which employs the inverse FFT. Notably, the Quantum Walk Sequencer is a sequencer that facilitates note-to-note transitions using a quantum random walk ¹¹. 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 ¹².

Kirke proposed the hybrid music generation system qGEN, which integrates a gated quantum computer

-
- 3 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).
 - 4 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).
 - 5 Weimer H.: Listen to Quantum Computer Music. <http://www.quantenblog.net/physics/quantum-computer-music>. *Physical Review A*, Vol.48, No.2, pp.1687--1690 (1993).
 - 6 Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing (1996).
 - 7 Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings of the 35th annual symposium on foundations of computer science, pp.124--134 (1994).
 - 8 Eduardo R. M. Bob C., et al.: QuTune Project. <https://iccmr-quantum.github.io/>.
 - 9 Miranda, E.R.: Quantum Computer: Hello, Music!. *Handbook of Artificial Intelligence for Music*, pp.963--994 (2021).
 - 10 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).
 - 11 Aharonov, Y., Davidovich, L., Zagury, N.: Quantum random walks. *Physical Review A*, Vol.48, No.2, pp.1687--1690 (1993).
 - 12 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).

and a quantum annealing machine ¹³. 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 ¹⁴. This approach involves producing melodies by connecting consecutive notes through quantum entanglement.

The efforts outlined in this section, which apply quantum computing to musical expression, have emerged in recent years, signifying the nascent development of this research domain.

Possibilities of Quantum Computing in Music Generation

In this chapter, we examine the potential of applying quantum computers to the domain of music generation. As previously discussed, numerous efforts have emerged in recent years, highlighting the growing interest in quantum music research.

A crucial characteristic of quantum computers is the superposition state of qubits. However, the superposition state collapses upon measurement, resulting in the qubit adopting either a 0 or 1 state, similar to classical bits. To solve a problem using a quantum computer, a quantum algorithm is designed to leverage superposition states, representing all possible input states, and increase the likelihood of obtaining the desired outcome through measurement. To date, there are few identified problems for which quantum computers can achieve results more rapidly than classical computers. Presently, algorithms targeting specific issues, such as factorization ⁷ and combinatorial optimization, are being proposed; however, these have not yet been implemented with practical accuracy or scale.

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. For tasks such as searching, employing an approach such as Grover’s algorithm ⁶ could facilitate the efficient retrieval of desired music. However, 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. This issue has been a long-standing consideration in music generation research, whether it is on a classical or quantum computer.

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. As a result, it is currently uncertain whether a quantum computer can produce better results than a classical computer in the task of melody generation.

13 Kirke, A.: Programming gate-based hardware quantum computers for music. *Physical Review A*, Vol.48, No.2, pp.1687--1690 (1993).

14 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).

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. Unlike the pseudo-random numbers generated by classical computers, the probabilistic nature of qubits in a superposition state ensures that true random numbers without any regularity or reproducibility can be generated. Existing music generation algorithms often rely on processes that utilize random numbers, making it potentially advantageous to incorporate quantum computers in this aspect. However, the precision of random numbers is not critical in music generation, so it is debatable whether introducing a quantum computer would be of great significance.

In this paper, we propose a melody generation algorithm that involves designing quantum circuits to replicate the note transitions observed in training data, using random number generation with a quantum computer. While a classical computer can describe this algorithm more concisely except for the random number generation aspect, we present an example of implementing it on a quantum computer. In this paper, we explore an alternative approach to conventional melody generation, namely, music generation using a quantum computer.

When considering the future application of quantum computers in music generation, the most likely scenario involves the use of hybrid algorithms with classical computers. For example, a hybrid algorithm could involve a classical computer processing data and a quantum computer narrowing down the results from all possible combinations. Many experts doubt that all current computers will be replaced by quantum computers in the future. Instead, it is believed that quantum computers will only be used to handle a portion of the processes currently performed by classical computers, particularly those that can be executed more efficiently using superpositioned states. Therefore, it is expected that quantum computers will only partially take over some processing tasks while the remaining tasks will still be performed by classical computers using the conventional approach.

Data Representation for Quantum Circuit Design

In this chapter, we will introduce the data representation for our melody generation method using a quantum computer. The proposed approach involves combining quantum gates to form a quantum circuit for melody generation using a genetic algorithm.

When designing quantum algorithms, it can be challenging to determine how to construct a quantum circuit that achieves the desired input-output relations. The greater the complexity of the quantum circuit, the more difficult its design becomes. For instance, designing a simple quantum circuit that generates the constituent notes of a C chord (i.e., C, E, and G) with equal probability is not a difficult task. On the other hand, designing quantum circuits becomes increasingly challenging when considering more complex conditions, such as generating notes with a subtle bias in probability distribution. Given these challenges, this paper proposes a method for automatically designing quantum circuits that can achieve specific input-output conditions for given notes. Our approach involves learning the composition of a quantum circuit that produces the desired output distribution through repeated input-output trials using various combinations of quantum gates. To achieve this, we employ a genetic algorithm.

To generate melodies using quantum circuits, it is essential to determine the appropriate data representation method. This includes representing both the notes and quantum gates numerically. Our proposed method aims to generate quantum circuits that produce the desired output distributions by combining various numerically represented quantum gates to achieve specific input-output relations.

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, each digit of a 3-bit binary number corresponds to a qubit. The problem is set up such that a 3-qubit quantum circuit generates the next note based on the input note.

Table 1 A binary representation of a note name.

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

Data Representation of Quantum Gates

There are various types of quantum gates that compose quantum circuits. In this paper, to realize melody generation in the simplest problem setting, we selected four basic types of gates and numerically represented 16 different gate placement patterns. These patterns include variations in which qubits the gates will act upon among the three 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. These 16 different gate arrangements, represented by 8-digit hexadecimal numbers, enable the representation of quantum circuits with 0 to 8 gate combinations. The numerical representation of the quantum gates and their corresponding gate arrangement patterns are shown in Table 2.

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 : (register 0)	3
X-axis rotation gate : (register 1)	4
X-axis rotation gate : (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
CX gate (register 2 to 1)	11
CCX gate (register 0,1 to 2)	12
CCX gate (register 0,2 to 1)	13
CCX gate (register 2,1 to 0)	14
no gate	15

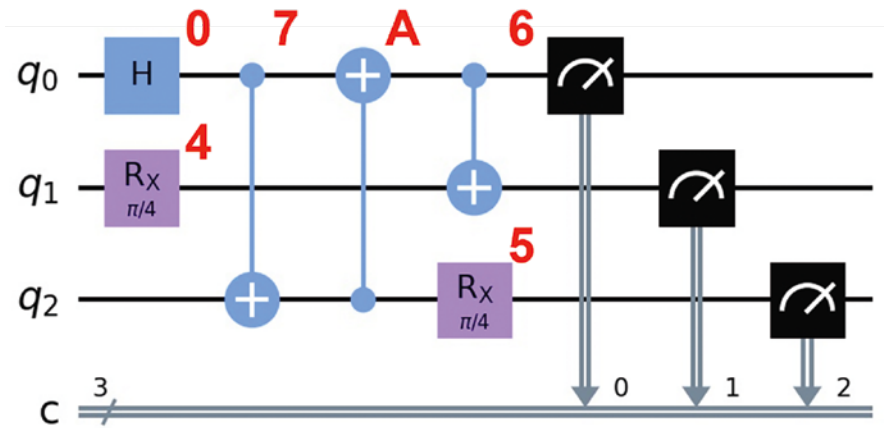


Figure 1 Example of quantum circuit represented by “04F7A6F5.”

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 Figure 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 in Figure 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.

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.

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 Measurement results (for 200 shots) when (C) is input to the quantum circuit shown in Figure 1.

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

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 Figure 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 Figure 1 without using the H gates.

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. We also investigated using the same quantum circuit for all input notes as another method (training B). The design process of each quantum circuit will be presented in the next chapter.

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 designing quantum circuits, and one example is to use a single H gate in every register to represent a superposition of all output combinations, resulting in a circuit with completely random outputs.

Manual determination of the gates in a quantum circuit is possible, but it can result in an inflexible quantum circuit design that produces fixed patterns of generated outcomes. Consequently, this paper explores a method that enables the flexible modification of quantum circuit designs by leveraging training data. Specifically, we prepare arbitrary melodies as training data and subsequently search for the optimal combination of quantum gates capable of reproducing 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 inherent in the training data.

A genetic algorithm is utilized as a means to explore combinations of quantum gates, with the objective of minimizing the discrepancy between the quantum circuit's output and the training data's distribution of note transition. It is important to note that the genetic algorithm is processed using a classical computer, while the quantum computer is responsible solely for generating subsequent notes based on input notes. This implies that our approach is also a hybrid method. Moreover, the learning processes are predominantly conducted using a quantum circuit simulator on a classical computer. The actual quantum computer is expected to be employed solely for generating melodies with the resulting quantum circuit. Although leveraging an actual quantum computer during the quantum circuit design process is feasible, the extensive number of trials necessary for learning renders it challenging to achieve learning within a realistic time frame, given the current state of quantum computing capabilities. It is anticipated that advancements in quantum computing technology will address these challenges in the future.

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 by employing a quantum circuit with a single Hadamard (H) gate inserted in the second register. As the actual training process entails a more intricate distribution of outputs, the quantum circuit design will be automated using a genetic algorithm, rather than being manually configured.

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

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 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 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.

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 previous section. 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 “FFFFFFF.” 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.

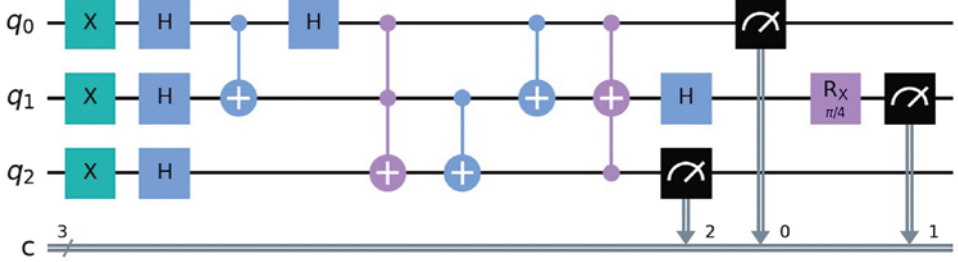


Figure 2 A quantum circuit designed by genetic algorithm (“60C86D14”), with the input note R (Training-A).

One of a quantum circuit designed by the genetic algorithm utilizing the training data (Table 4) is depicted in Figure 2. Given that the quantum circuit in Figure 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 transitions to $|111\rangle$, 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 of Figure 2 contains certain redundant gate arrangements, such as consecutive utilization of the CX and CCX gates, the output from this circuit exhibits a distribution closely approximating the training data (bottom row of Table 4). The output result will be described in the next section.

In Training-A, quantum circuits are tailored for each input note, resulting in a total of eight distinct circuits designed using the genetic algorithm. The design results for all quantum circuits, excluding the input note R, are provided in an appendix at the end of this paper.

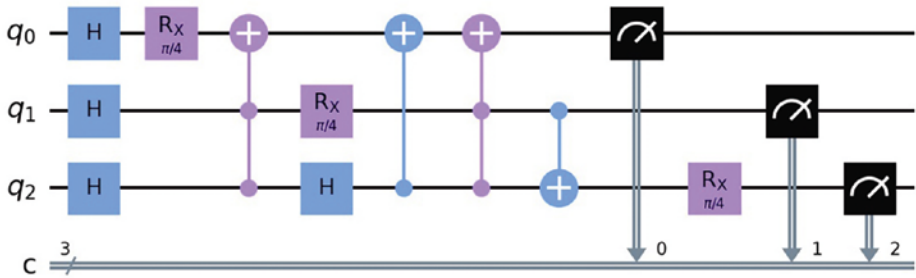


Figure 3 A quantum circuit designed by genetic algorithm (“3D24ADB5”), for all input/output note combinations (Training-B). X gates for distinguishing the types of input notes are omitted here.

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 Figure 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.

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 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.

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 (Figure 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

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 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.

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 of reproducing subsequent note is defined as follows:

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

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 R 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 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}

A comparison of the error rate in reproducing subsequent notes using each circuit/computer is shown in Table 7. The smaller the value of R , the higher the reproducibility of the training data. The classical computer achieved high reproduction accuracy because the transition probability distribution of the training data was directly used to generate the random numbers. On the other hand, in the case of quantum circuits, the error rate of reproduction was lower when using different quantum circuits designed for each input note, i.e. Training-A setting. This indicates that different quantum circuits for different input conditions are more likely to obtain the desired output distribution. The error rate of the quantum circuits (for each input note) in this result is approximately 6-7 times higher than that of the classical computer. However, 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 (and 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.

Melody Generation Results

We employed the quantum circuit, which was designed based on the methodology outlined above, for the purpose of generating musical melodies. The melody generation process begins with the selection of the first note, followed by obtaining the output when this first note is input into the quantum circuit specifically designed for the input note. Subsequently, 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.

Tables 8 and 9 present examples of the generated results, with four trials each, for generating a melody spanning four measures (comprising 16 notes in total), with the initial notes set to C and G, respectively. Notably, the subsequent notes are determined probabilistically, resulting in distinct output melodies for each execution. These results were obtained using the Qasm Simulator, which is integrated within the Qiskit framework.

Table 8 Example of generated melodies (note name sequences) using 8 different trained quantum circuits (Training-A).

first note	Generated melody (note sequence)															
C	C	R	C	C	C	R	E	G	A	G	G	E	D	D	C	D
C	C	D	E	D	E	C	D	E	E	R	G	F	E	E	E	D
G	G	E	D	C	G	E	R	C	D	D	D	E	R	E	D	D
G	G	A	G	G	F	E	D	C	R	G	E	G	F	F	E	G

Table 9 Example of generated melodies (note name sequences) using a single quantum circuit for all input notes (Training-B).

first note	Generated melody (note sequence)															
C	C	A	A	R	R	G	G	G	F	E	D	R	D	D	C	C
C	C	R	B	E	R	A	G	E	D	D	D	F	A	A	G	R
G	G	E	B	A	A	E	C	R	G	G	F	F	E	B	F	D
G	G	A	B	F	F	C	G	G	G	G	F	G	F	F	F	D

In certain instances, the generated melodies encompassed transitions to note B, which were absent from the training data. Conversely, some cases exhibited note transitions that were present in the training data, effectively capturing the ambiance of the trained melody.

In the next stage, we generated a melody employing an actual quantum computer. The scores of melodies generated by the quantum computer are depicted in Figure 4 and 5. Two 4-measure melodies were generated for each quantum circuit design pattern (Training-A and Training-B), with the initial notes set to C and G. The quantum computer utilized in this study was the `ibm_bogota`, which can operate up to five qubits and is provided by IBM on the cloud through IBM Q. 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 Figure 4 bottom) and Table 6 (D→G in Figure 5 top).

Melody generated with C as the first note



Melody generated with G as the first note



Figure 4 Examples of melody generation utilizing an actual quantum computer, employing eight different trained quantum circuits for each input note (Training-A).

Melody generated with C as the first note



Melody generated with G as the first note



Figure 5 Examples of melody generation utilizing an actual quantum computer, 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.

Conclusions

In this paper, we explored the potential application of quantum computers in music generation and proposed a method for designing quantum circuits for melody generation using a genetic algorithm. We compared the accuracy of reproducing subsequent notes between quantum and classical computers. The results of new melodies generated using quantum circuits trained on specific musical pieces were presented. Although a similar melody generation can be achieved with a classical computer, the distinct difference lies in the randomness of the outcome. The current implementation does not fully take advantage of the unique characteristics of a quantum computer. 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. We successfully represented the input/output relationship of notes using quantum circuits. This finding suggests that various music generation algorithms currently implemented in 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 technology is currently in the early stages of development. It is anticipated that 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.

Acknowledgement

This work was partially supported by JSPS KAKENHI Grant Number JP19K20301.

Appendix: Examples of Quantum Circuits Designed Using the Proposed Method

The quantum circuit, designed by the method detailed in this paper under the Training-A setting, is presented as follows (refer to Figure 2 for the input note R).

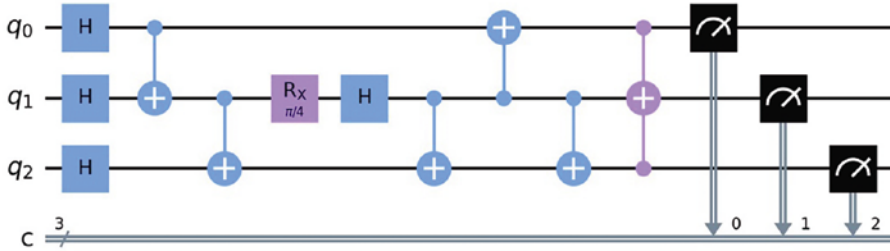


Figure A-1. Quantum circuit (“6841898D”) designed by proposed method (input note C).

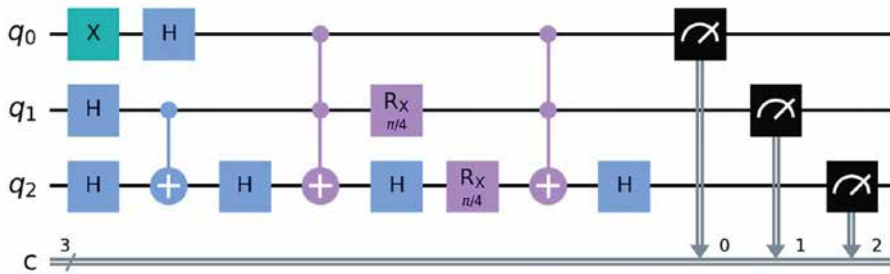


Figure A-2. Quantum circuit (“B2C245C2”) designed by proposed method (input note D)

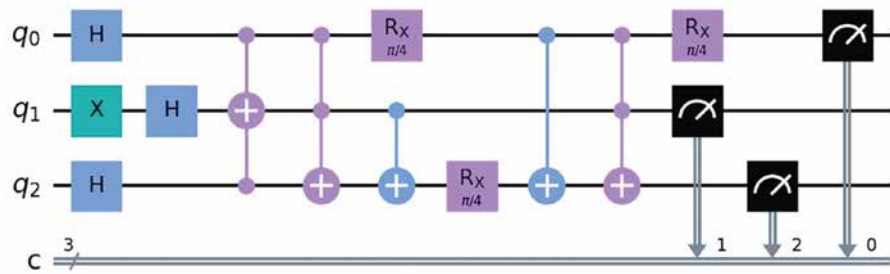


Figure A-3. Quantum circuit (“DCB537C3”) designed by proposed method (input note E).

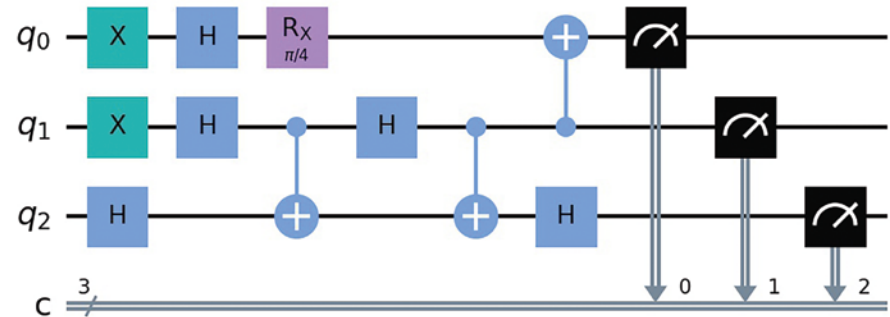


Figure A-4. Quantum circuit (“83FF1892”) designed by proposed method (input note F).

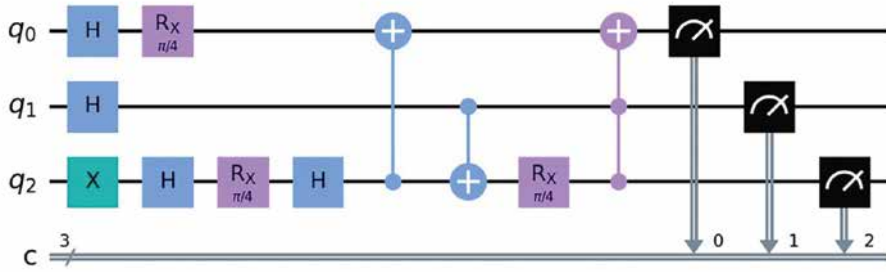


Figure A-5. Quantum circuit (“3F52AB5E”) designed by proposed method (input note G).

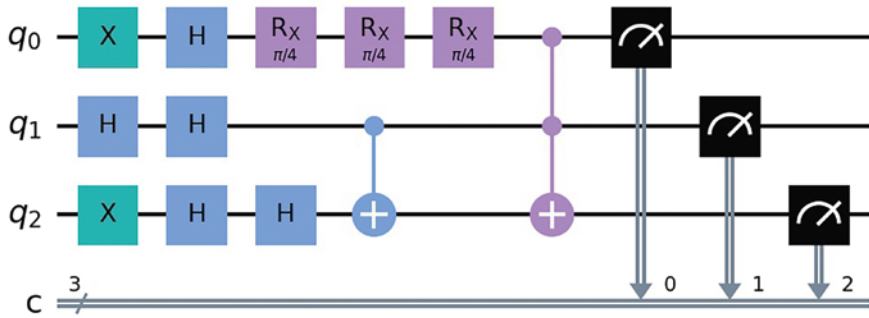


Figure A-6. Quantum circuit (“13233FBC”) designed by proposed method (input note A).

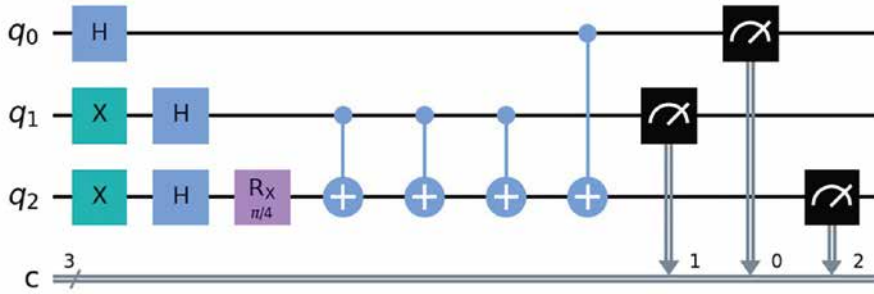


Figure A-7. Quantum circuit (“5FBBFBF7”) designed by proposed method (input note B).