

2022-09-16

Teaching Qubits to Sing: Mission Impossible?

Miranda, E

<http://hdl.handle.net/10026.1/19633>

International Journal of Unconventional Computing
Old City Publishing

All content in PEARL is protected by copyright law. Author manuscripts are made available in accordance with publisher policies. Please cite only the published version using the details provided on the item record or document. In the absence of an open licence (e.g. Creative Commons), permissions for further reuse of content should be sought from the publisher or author.

Teaching Qubits to Sing: Mission Impossible?

Eduardo Reck Miranda Brian N. Siegelwax

Interdisciplinary Centre for Computer Music Research (ICCMR),
Faculty of Arts, Humanities and Business,
University of Plymouth, Plymouth, PL4 8AA,
United Kingdom
eduardo.miranda@plymouth.ac.uk, bsiegelwax@gmail.com

August 9, 2022

Abstract

This paper introduces QuSing, a system that learns to sing new tunes by listening to examples. QuSing extracts sequencing rules from input music and uses these rules to generate new tunes, which are sung by a vocal synthesiser. We developed a method to represent rules for musical composition as quantum circuits. We claim that such musical rules are *quantum native*: they are naturally encodable in the amplitudes of quantum states. To evaluate a rule to generate a subsequent event, the system builds the respective quantum circuit dynamically and measures it. After a brief discussion about the vocal synthesis methods that we have been experimenting with, the paper introduces our novel generative music method through a practical example. The paper shows some experiments and concludes with a discussion about harnessing the system's creative potential. Accompanying materials are available in an Appendix. Audio recordings of the musical examples and programming code are available: <https://github.com/iccmr-quantum/QuSing>.

1 Introduction

Research and development in computer music and professional usages have been progressing in tandem with Computer Science since the invention of the computer. Musicians started experimenting with computers far before the emergence of the vast majority of scientific, industrial and commercial computing applications in existence today. For instance, in the 1940s, researchers at Australia's Council for Scientific and Industrial Research (CSIR) installed a loudspeaker on their Mk1 computer to track the progress of programs using sound. Subsequently, Geoff Hill, a mathematician with a musical background, programmed this machine to play back a tune in 1951 [12].

The first uses of computers in music were for composition. The great majority of computer music pioneers were composers interested in developing innovative approaches to composition [20, 27, 30]. Nowadays, computing technology is omnipresent in almost every aspect of the music industry [5, 53]. Therefore, ever-evolving quantum computing technologies will continue to impact how we create, perform, listen and commercialize music in time to come. In

the newborn field of *Quantum Computer Music* [19, 23, 28, 33, 35], researchers and practitioners are exploring ways to leverage the quantum-mechanical nature of quantum computing to compose, perform, analyse and synthesise music and sound.

Classical computers manipulate information represented in terms of binary digits, each of which can value 1 or 0. They work with microprocessors made up of billions of tiny switches that are activated by electric signals. Values 1 and 0 reflect the on and off states of the switches.

In contrast, a quantum computer deals with information in terms of quantum bits, or qubits. Qubits operate at the subatomic level. Therefore, they are subject to the laws of quantum physics.

At the subatomic level, a quantum object does not exist in a determined state. Its state is unknown until one observes it. Before it is observed, a quantum object is said to behave like a wave. But when it is observed it becomes a particle. This phenomenon is referred to as wave-particle duality.

Quantum systems are described in terms of wave functions. A wave function expresses the state of a quantum system as the sum of the possible states that it may fall into when it is observed. Each possible component of a wave function, which is also a wave, is scaled by a coefficient reflecting its relative weight. That is, some states might be more likely than others. Metaphorically, think of a quantum system as the spectrum of a musical sound, where the different amplitudes of its various wave components give its unique timbre. As with sound waves, quantum wave components interfere with one another, constructively and destructively. In quantum mechanics, the interfering waves are said to be coherent. The act of observing waves decoheres them. Again metaphorically, it is as if when listening to a musical sound one would perceive only a single spectral component; probably the one with the highest energy, but not necessarily so.

Qubits are special because of the wave-particle duality. Qubits can be in an indeterminate state, represented by a wave function until they are read out. This is known as superposition. A good part of the art of programming a quantum computer involves manipulating qubits to perform operations while they are in such an indeterminate state. This makes quantum computing fundamentally different from digital computing.

An introduction to the nuts and bolts of quantum computing is beyond the scope of this paper. This can be found in [4, 16, 44, 47, 55].

This paper presents QuSing, a system that learns to sing tunes from given examples. The system extracts compositional rules from given pieces of music and uses these rules to generate new tunes rendered with a vocal synthesiser (Figure 3). The new vocal tunes resemble the original music and the system provides the means to vary the degree of resemblance. In a nutshell, the compositional rules are extracted classically, so to speak, but the engine that processes the rules to generate the new tunes are processed quantumly.

A preliminary attempt at using quantum computing to control a vocal synthesiser was presented in [33]. In that case, the parameters for vocal synthesis were derived from bitstrings produced using a quantum 'hyper-dice' consisting of nine qubits. Albeit simplistic, this work paved the way for the development of QuSing.

The paper begins by presenting the vocal synthesisers that are used to render the singing, followed by reviewing the notion of musical composition with transition rules. Then, it introduces the quantum computing method that we invented to generate the new tunes. Next, it walks through a detailed example illustrating how the system works. The paper concludes with a discussion and some final remarks, which includes work that is in progress. The Appendices

contain accompanying materials. Audio recordings of the musical examples are available on SoundClick [38] and programming code can be found from QuSing's GitHub repository [42].

2 Vocal Synthesis

QuSing has two options for synthesising the singing. One uses a bespoke implementation of a well-known formant synthesis method, often referred to as Klatt synthesis [24]. The other uses a commercially available system called Vocaloid [22], which is based on a method known as concatenative synthesis [48]¹.

The frequency spectrum of a vocal sound has the appearance of a pattern of 'hills and valleys', technically called *formants* (Figure 1). When speaking or singing, air streams are forced upwards through the trachea from the lungs. Technically, such streams are comparable to audio signals and are referred to as *excitation signals*.

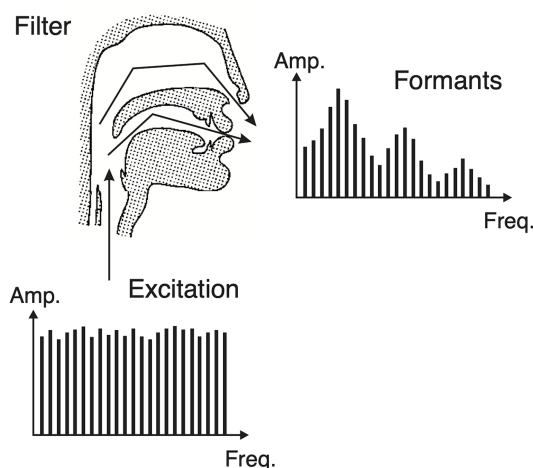


Figure 1: When singing, excitation is forced upwards through the trachea from the lungs. On its journey through the vocal tract, the excitation signal is transformed, producing formants.

At the base of the larynx, the vocal cords are folded inwards from each side, leaving a variable tension and a slit-like separation, controlled by muscles. In normal breathing, the cords are held apart to permit the free flow of air. In singing or speaking, the cords are brought close together and tensed. The forcing of the excitation through the vocal cords in this state sets them into vibration. As a result, the excitation signals are modulated at the vibration frequency of the vocal cords. This motion determines the pitch of a vocal sound.

The vocal system can be thought of as a resonating structure in the form of a pipe (Figure 1), from the vocal cords to the lips, plus a side branch leading to the nose. On its journey through the vocal tract, the excitation is transformed (that is, filtered) by the resonance of the vocal 'pipe'. Components that are close to the resonance frequencies of the tract are transmitted with high amplitude, while those which lie far from the resonance frequencies are suppressed,

¹The accompanying sound examples for this paper were produced using the latter.

hence the hills and valleys pattern in the frequency spectra of vocal sounds. The art of singing lies in controlling the energy of the excitation, the tension of the vocal folds and the shape of the vocal tract to produce the desired tones.

Traditionally, a Klatt formant synthesiser is a *subtractive synthesiser* [18] consisting of an excitation module and a resonator module. For more on Klatt's method see also [1, 24, 45]. QuSing's implementation of Klatt's method deploys generators of excitation signals (e.g., band-limited noise and pulse generators) and a bank of band-pass filters tunable to produce formants. Originally, this method was developed for producing speech. We adapted it for synthesising singing voice.

Alternatively, some systems focus on replicating the actual sounds rather than modelling the vocal system. Normally, such systems hold a database of recorded basic building blocks (e.g., phonemes, onomatopoeias, vowels), which are picked and concatenated to form words, tunes, and so on. Some of them deploy a technique referred to as analysis-resynthesis [46]. Think of this as something like applying a Fourier Transform [43] to analyse the sound and then doing an inverse Fourier Transform to re-synthesise it. The system analyses those building blocks to extract spectral features and store them. Then, rather than necessarily concatenating the original sounds per se, the system concatenates the respective analysis information, which is then used to synthesise the utterance. In this case, the system can manipulate the analysed information to achieve particular effects, such as changing the prosody of the utterance, pitch, vocal timbre, and more [26, 51]. Essentially, the Vocaloid system is a *concatenative synthesiser*, which uses analysis-resynthesis and machine-learning to process vocal fragments extracted from recordings of humans singing [54].

In both cases, the decoded musical events (Figure 3) are translated onto control parameters for the respective synthesisers. In the simplest case scenarios, which are the ones reported in this paper, they control the pitches and lengths of the excitation signals (Klatt synthesiser) or the pitches and lengths of the concatenated segments (Vocaloid).

3 Composing with Transition Rules

Humans possess an irresistible urge to produce and appreciate sound arrangements beyond the mere purposes of signalling or linguistic communication. It seems that we do this for no specific purpose other than to enjoy it. This is an intriguing evolutionary trait that differentiates us from other animals [8]. But what is music?

As a working definition, let us establish that *music is sounds organised in space and time*. When we listen to music, our brain has certain expectations; for instance, it expects some order in the auditory stimuli. In music theory, this is generally referred to as *musical form*.

Psychology teaches us that our auditory system employs mental schemes to make sense of streams of sounds [31]. However, there is no agreement about which of such mental schemes are genetically hard-wired in our brains and which ones evolve culturally as we grow up. This suggests that there is no good or bad music; it depends on culture, individual taste, and so on. But we should not need to worry about this debate here. What is important is to concur that music creators organise sounds according to some criteria. Thus, for computers to create music one needs to endow them with such criteria; to glance over different approaches for doing this, please refer to [7, 11, 37, 39]. One such approach is to program the computer with (a) rules for sequencing musical events and (b) the ability to use those rules.

As an example, let us consider a given set of eight musical pitches as follows:

$$\{C_3, D_3, E_3, F\#_3, G\#_3, A\#_3, C_4, D_4\}$$

To create musical forms with those pitches, we need some rules governing how they can be sequenced; e.g., some pitches might not be allowed to follow a certain pitch, or some might have priority to follow some others, and so on. Let us define a few transition rules, as follows:

- $C_3 \Rightarrow D_3(25\%) \vee G\#_3(25\%) \vee C_4(25\%) \vee D_4(25\%)$
- $D_3 \Rightarrow C_3(30\%) \vee E_3(70\%)$
- $E_3 \Rightarrow D_3(25\%) \vee F\#_3(25\%) \vee A\#_3(25\%) \vee C_4(5\%) \vee D_4(20\%)$
- $F\#_3 \Rightarrow E_3(100\%)$
- $G\#_3 \Rightarrow C_3(30\%) \vee A\#_3(70\%)$
- $A\#_3 \Rightarrow E_3(33\%) \vee G\#_3(33\%) \vee C_4(34\%)$
- $C_4 \Rightarrow C_3(30\%) \vee A\#_3(70\%)$
- $D_4 \Rightarrow C_3(20\%) \vee E_3(80\%)$

The symbol “ \vee ” stands for “or” and the percentage figure in parenthesis next to the notes is their weight coefficient, expressed here in terms of probability of occurrence. For instance, the second rule states that given a pitch D_3 , only C_3 or E_3 can follow it, but E_3 has a higher priority to occur. In other words, the rule is saying that given a pitch D_3 , there is a 30% chance that it would be followed by a C_3 and a 70% chance that it would be followed by an E_3 . For didactic purposes, with simple rule systems like the example above, it is often useful to visualise it as a *transition table* (Figure 1).

	C_3	D_3	E_3	$F\#_3$	$G\#_3$	$A\#_3$	C_4	D_4
C_3		25%			25%		25%	25%
D_3	30%		70%					
E_3		25%		25%		25%	5%	20%
$F\#_3$			100%					
$G\#_3$	30%					70%		
$A\#_3$			33%		33%		34%	
C_4	30%					70%		
D_4	20%		80%					

Table 1: Transition table representing the rules.

It should be said that such rules do not necessarily need to consider just one previous or just one subsequent event. There could be rules like: $\{D_3 \rightarrow C_3\} \Rightarrow E_3(30\%) \vee G\#_3(70\%)$, or $\{D_3 \rightarrow C_3 \rightarrow D_3\} \Rightarrow \{E_3 \rightarrow C_4\}(70\%) \vee G\#_3(30\%)$, and so on.

Given a set of rules, it is crucial to devise a method to compute them to generate sequences. There are many ways of doing this with ‘classical’ computing methods, which will not be discussed here [3, 17, 49]. Rather, we are interested in exploring ways of doing it quantumly. In section 4 we introduce the method that we invented for this and in section 5 we walk through an illustrative example. We argue and demonstrate that the rules for musical composition, of the type presented here, are *quantum native*. They are naturally encodable in the amplitudes of quantum states.

4 Computing Transition Rules Quantumly

There have been some previous attempts at designing quantum algorithms to generate music with transition rules.

Weaver proposed a system whereby a 4×4 matrix represents transitions rules (similar to those shown in section 3) for a set of four notes. He designed a quantum circuit using two qubits and rotation operator gates. The probabilities in the matrix are converted into angles for the rotation gates [52]. The caveat of this system is that it is limited to a set of four notes. But to a certain extent, the method that we propose below builds upon Weaver's idea. Our method scales up to any number of notes.

Another method is the *Basak-Miranda algorithm* [32], which leverages a property of quantum mechanics known as constructive and destructive interference to compute the rules. It is based on the well-known Grover's algorithm [10], which has become a favoured example to demonstrate the advantage of quantum computing for searching for information in databases. However, the Basak-Miranda algorithm is limited to using equal weight distribution between the possible next notes. For instance, two possible notes would have a $50\% \times 50\%$ distribution between them by default.

A different approach uses quantum annealing, which is an alternative model of quantum computation [2, 9]. Quantum annealing is suitable for propositional (Boolean) satisfiability problems² and other combinatorial searching problems. The selection of suitable paths in a transition rule can be modelled as a satisfiability problem. This approach works well but requires special hardware, which jeopardises its compatibility for integration with potentially more generic gate-based quantum computing systems. A comparison between gate-based and quantum annealing models of computation is beyond the scope of this paper; please refer to [10, 13, 50].

Let us introduce our method through an example using the same pitch set and rules used in section 3.

Each time the system needs to evaluate a transition rule it builds and measures a quantum circuit that is specific to the rule in question. The circuit is built in such a way that the wave function that defines the state of the quantum system represents the probability distribution of the rule in question. That is, the measurement is likely to collapse the qubits to one of the allowed options for the next event, considering the probability weights.

First of all, we assign binary codes to the pitches, as follows:

- $C_3 \Rightarrow 000$
- $D_3 \Rightarrow 001$
- $E_3 \Rightarrow 010$
- $F\#_3 \Rightarrow 011$
- $G\#_3 \Rightarrow 100$
- $A\#_3 \Rightarrow 101$
- $C_4 \Rightarrow 110$
- $D_4 \Rightarrow 111$

²A propositional satisfiability problem, often called SAT, is the problem of determining whether a set of logic sentences is satisfiable.

Now, let us consider the rule $C_4 \Rightarrow C_3(30\%) \vee A\#_3(70\%)$. In terms of the binary representation, this rule is expressed as $110 \Rightarrow 000(30\%) \vee 101(70\%)$. In this case, the system builds the circuit shown in Figure 2. As the set has eight pitches, the circuit needs only three qubits to represent every possible solution.

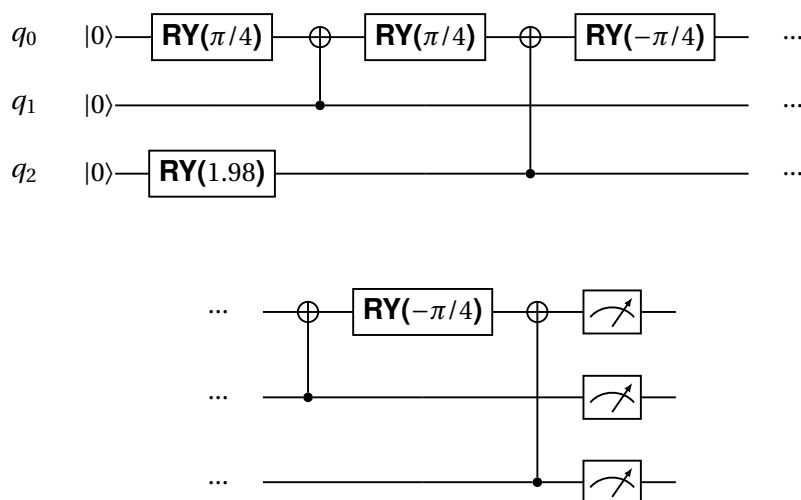


Figure 2: Circuit for the rule $100 \Rightarrow 000(30\%) \vee 101(70\%)$.

Upon measurement, the circuit (Figure 2) should output either 000 with 30% probability or 101 with 70%. (The ordering of the qubits are $q_2 q_1 q_0$.)

5 The QuSing System

A flow diagram depicting a bird’s eye view of QuSing is shown in Figure 3. This section walks through each stage using a simple example. More examples are given in the Appendix. The quantum computing components of QuSing were implemented in Qiskit³ and we ran the experiments and demonstrations discussed in this paper using IBM Quantum’s resources [21].

The system has two main phases, highlighted within blue and red dashed boxes: a machine learning phase (blue) and a generative one (red). In a nutshell, in the machine learning phase, the system ‘listens’ to one or more musical tunes and extracts probabilistic rules governing their structure. Then, in the generative phase, it uses these rules to generate new tunes and ‘sings’ them. Once the system has learned the rules, it can generate as many new songs, of virtually any length, as required. The learning is done classically and the generation of new music is done quantumly.

³Qiskit is an open-source software development kit (SDK) for programming quantum computers using Python.

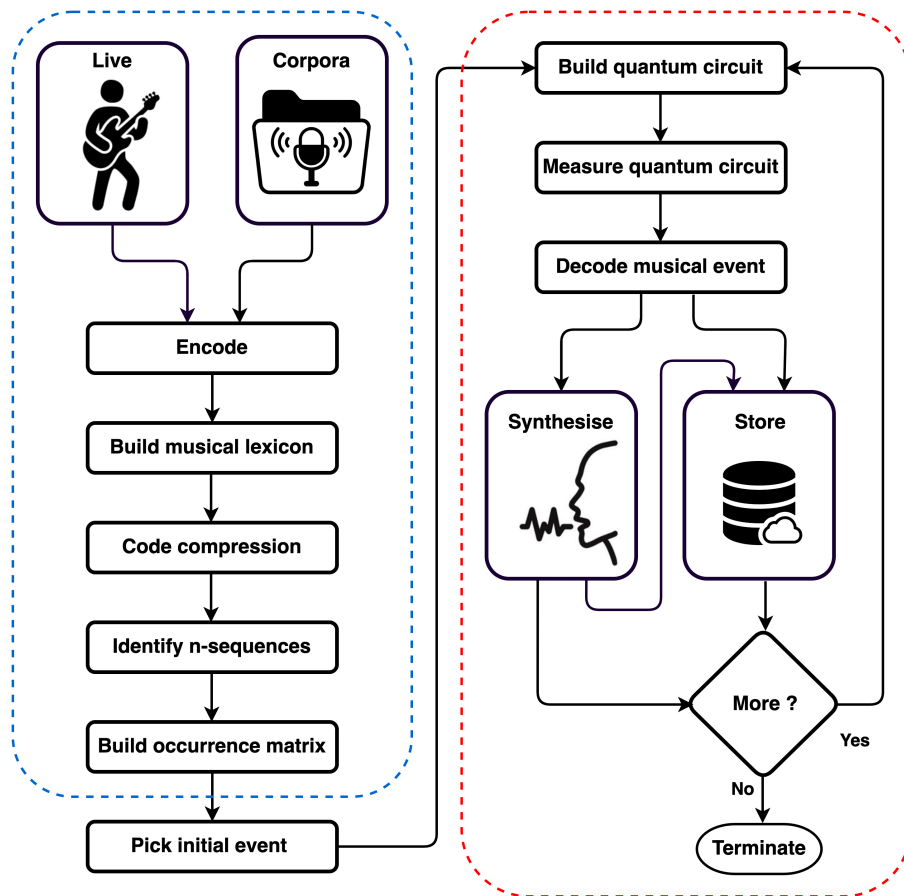


Figure 3: System's flow diagram.

5.1 Music input

QuSing accepts two types of inputs: live music input and a MIDI⁴ music file (or a set thereof). The live input can be from a MIDI instrument or audio, through a microphone or pickup. If audio is used, then it needs to be converted to MIDI for the Encode stage⁵

The system only accepts monophonic music; i.e., single-note tunes rather than more than one note sounding simultaneous, or chords. As an example, let us input the MIDI file of an extract of the *Mission: Impossible*⁶ theme, shown in Figure 4.

⁴MIDI is an acronym that stands for Musical Instrument Digital Interface. It is a technical standard that describes a communications protocol, to connect electronic musical instruments, computers, and related audio devices for playing, editing, and recording music.

⁵It would be possible to process audio in the Encode stage. But this would require significant effort. It is convenient to convert audio to MIDI first, as there are audio-to-MIDI technologies readily available; e.g., [6].

⁶This is an espionage television series aired on USA TV in the 1960s and 1970s. Several films followed casting Hollywood stars such as Tom Cruise and Henry Cavill.



Figure 4: An extract from the *Mission: Impossible's* theme.

5.2 Encode

Firstly, the system needs to convert the input's MIDI codes to the system's own bespoke representation, which is more efficient than MIDI for our purposes.

The notes and pauses are viewed as compounds formed by a pitch (or lack of it, in the case of a pause) and a duration. For instance, in Figure 4, the first and the fourth notes are identical: 'B₄ quaver'. But the ninth and the eleventh are not: one is a 'C₄ dotted minim' and the other is a 'C₄ crotchet'. To avoid confusion with standard music theory, from now on, a note or pause will be referred to as an *event*.

An event is encoded using a string of nine binary digits (Figure 5). The first five digits encode its pitch and the subsequent four encode its duration, that is, a rhythmic figure, such as minim, crotchet, quaver, and so on. Therefore, the system can process up to 32 distinct pitches (one of which is silence) and up to 16 distinct durations in a piece of music.

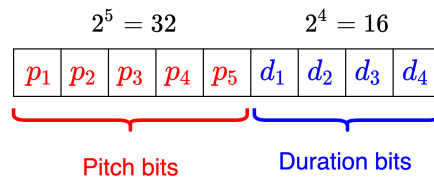


Figure 5: Scheme for encoding the events of a musical sequence.

For the input example in Figure 4, the system identified six pitches and four rhythmic figures, and calculated the corresponding 5-bit and 4-bit codes shown in Tables 2 and 3, respectively.

Note	MIDI	5-bit code
Silence	0	00000
B ₃	58	00001
C ₄	60	00010
C ₄ [#]	61	00011
D ₄	62	00100
G ₄	67	00101
B ₄	70	00110

Table 2: Bit codes for pitches. MIDI note number equal to 0 corresponds to silence; i.e., no pitch.

Duration	Ticks	4-bit code
quaver	480	0000
crotchet	960	0001
minim + crotchet	2400	0010
minim + quaver	2880	0011

Table 3: Bit codes for durations. Ticks are expressed in terms of the PPQN (Pulses per Quarter Note) time resolution of the MIDI input; e.g., a crotchet is set to 960 ticks.

In total there are 12 events in our input tune, represented as shown in Eq. 1. Let us refer to this as a training set \mathcal{T} . If there were more than one input file, then they would be organised as sub-sets of the overall training set \mathcal{T} .

$$\mathcal{T} = \{001100000, 001010000, 001000011, 001100000, 001010000, 000110011, 001100000, 001010000, 000100011, 000010000, 000100001, 000000010\} \quad (1)$$

5.3 Musical lexicon

The next step builds a lexicon of unique events in \mathcal{T} . In practice, the system removes all repetitions of identical events. For this example, the lexicon \mathcal{L} contains eight distinctive events, as shown in Eq. 2. Figure 6 highlights them on the score. (For now, ignore the binary codes written below them. These will be clarified below.)

$$\mathcal{L} = \{001100000, 001010000, 001000011, 000110011, 000100011, 000010000, 000100001, 000000010\} \quad (2)$$

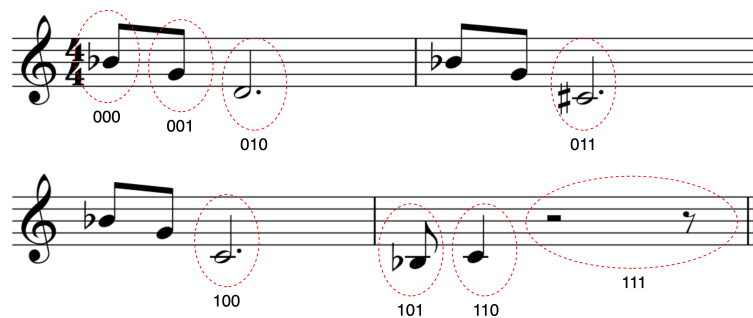


Figure 6: The input tune has eight distinct events, highlighted in red.

5.4 Code compression

At this stage, the system attempts to optimise, or compress, the lexicon, by relabelling its elements with, possibly, shorter binary codes. First, it estimates the number of bits needed to generate the new binary codes and then proceeds with the relabelling accordingly.

Code compression is an important step because the number of bits needed for the relabelling defines the number of *qubits* that will be needed to build quantum circuits in the generative phase. In our example, three bits are sufficient to represent the eight events of the lexicon. Thus, a reduction from nine to three bits, as follows (also shown in Figure 6):

- 001100000 \Rightarrow 000
- 001010000 \Rightarrow 001
- 001000011 \Rightarrow 010
- 000110011 \Rightarrow 011
- 000100011 \Rightarrow 100
- 000010000 \Rightarrow 101
- 000100001 \Rightarrow 110
- 000000010 \Rightarrow 111

5.5 Identify sequences and build an occurrence matrix

Next, the system builds an occurrence matrix with the number of times an event E_{t+1} (listed on the top row) follows another one E_t (listed on the first column on the left side). Figure 7 shows the occurrence matrix for the tune in Figure 4. For instance, the event ‘G₄ quaver’ (001) followed the event ‘B_{b4} quaver’ (000) three times. QuSing also considers E_t as a sequence of n events; e.g., rule {000 001} \Rightarrow 010(33%) \vee 011(33%) \vee 100(34%).

	000	001	010	011	100	101	110	111
000		3						
001			1	1	1			
010	1							
011	1							
100						1		
101							1	
110								1
111								

Figure 7: Occurrence matrix for the tune in Figure 4.

Recall from section 4 above that the quantum circuit to compute the next note needs the occurrence values specified in terms of amplitudes. Therefore, the numbers of occurrences in the matrix need to be converted into amplitudes. The converted matrix is shown in Figure 8. Note that the last row was removed because event 111 has no successor. Should this event be produced during a generative process (rule 110 \Rightarrow 111(100%)), then QuSing will add 111 to the new composition and generate the next event (pseudo-)randomly.

	000	001	010	011	100	101	110	111
000	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
001	0.0	0.0	0.58	0.58	0.58	0.0	0.0	0.0
010	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
011	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
100	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
101	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
110	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

Figure 8: Occurrence matrix with amplitudes, highlighted in red. The last row of matrix was removed.

5.6 Pick an initial event and initiate the generative process

Now, the system picks an event E_t to begin the generative process. By default, this is normally picked (pseudo-)randomly. But there is the option to start with the same beginning as the input piece (or from one of the input pieces, in case of using a corpus). For this example, we chose to begin with the input's first event: 'B_b₄ quaver' (000) (Figure 9). The system is now ready to initiate the generative process.



Figure 9: The initial note.

5.7 Build quantum circuit and measure

According to our occurrence matrix (Figure 8), only event 001 can follow event 000. The respective amplitude for 001 is equal to 1.0, which means that there is a 100% probability that 000 will be followed by 001 (Figure 11). The quantum circuit⁷ to compute this transition is rather simple (Figure 10): it should always measure $q_2 = 0$, $q_1 = 0$, $q_0 = 1$, that is 001. In practice, as we are dealing with a small occurrence matrix in this example, here we simply skip the quantum processing altogether, and retrieve the only possible choice classically; see discussion in section 6.

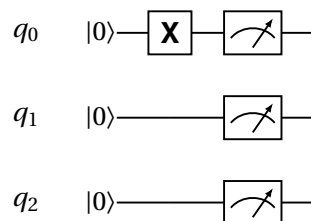


Figure 10: Circuit for the transition $000 \Rightarrow [0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]$.

⁷In our implementation, we used Qiskit's `initialize()` function to build the circuit.



Figure 11: The resulting second note is appended to the sequence.

At this point, the result $E_{t+1} = 001$ is appended into the new tune, E becomes equal to 001, and the system moves on to generate the next event.

According to our occurrence matrix (Figure 8), three events have equal probability of 33% ($0.58^2 \times 100 = 33$) to follow 001: 010, 011 and 100. In this case, the system builds the circuit shown in Figure 12. The respective measurement is then rendered into music and the process continues accordingly. In this example, the measurement returned 011, which corresponds to the event 'C#₄ dotted minim' (Figure 13).

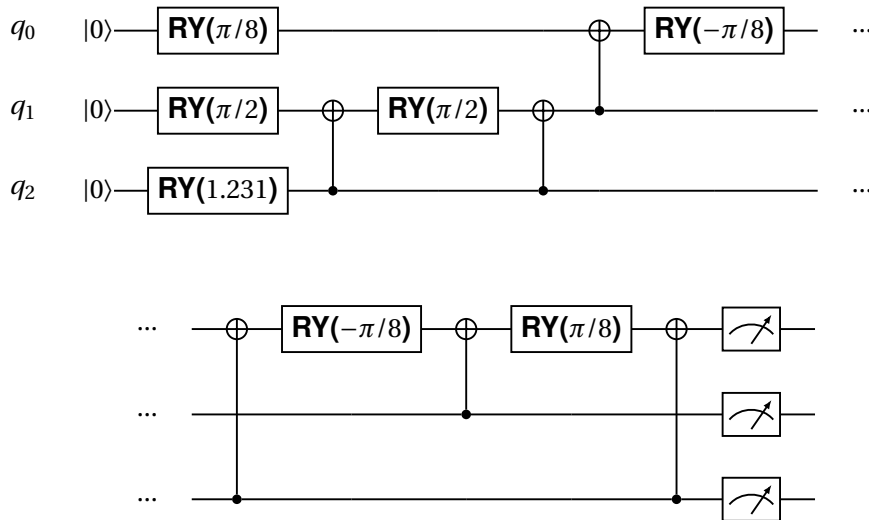


Figure 12: Circuit for the transition $001 \Rightarrow [0.0, 0.0, 0.58, 0.58, 0.58, 0.0, 0.0, 0.0]$.



Figure 13: The resulting third note is appended to the sequence.

5.8 Decode musical event and synthesise singing

Figure 14 shows the resulting tune after five generative iterations; see Figure 16 and the Appendix for more examples. Once a measurement is obtained, the value is used to retrieve the respective 'uncompressed' code (see section 5.4), which is then decoded according to Tables 2 and 3, respectively.



Figure 14: Generated output after five generative cycles.

The decoded event is either synthesised on the spot or appended into a MIDI file, which can be synthesised in batch mode after the process is terminated, or both. The synthesised tune is also stored either way.



Figure 15: *Mission: Impossible* theme.

Figure 15 shows a longer version of the *Mission: Impossible* theme input and a respective

QuSing-generated version is presented in Figure 16. This was generated with IBM Quantum’s `ibmq_lima` backend, which is a 5-qubit superconductor processor. Here we asked QuSing to produce 50 events, with one shot per each event⁸.



Figure 16: QuSing-generated tune, with 50 rounds, 1 shot each.

6 Discussion

We are interested in developing quantum computing systems for musical composition that supports the user’s creative process by enabling experiments with different settings to produce varied outcomes [34]. In this section, we examine QuSing’s behaviour under different conditions and discuss how this may be harnessed for musical experimentation.

6.1 Embracing noise

Currently, quantum computers are crippled by noise, or decoherence [44]: a quantum mechanics phenomenon that leads to processing errors. Coherence time is the length of time qubits can hold quantum information. When qubits are disrupted by external interference, such as temperature changes or stray electromagnetic fields, information about the state of the qubits is destroyed. This can ruin the ability to exploit quantum mechanics for computation. Longer coherence times enable more quantum operations to be utilised before this occurs. Significant work is being conducted by the research community to mitigate decoherence problems. For instance, circuit optimisation techniques are aimed at reducing the number of gates to be executed on the qubits, thus reducing the time they are required to remain coherent.

For this project, we used Quantinuum’s TKET optimisation tool [40] and Qiskit’s Pass Manager for circuit optimization [41] to make the circuits as small and efficient as possible. We

⁸Number of shots means how many times a circuit is run to get a probability distribution of results; see section 6 Discussion.

then used the Python package `mapomatic` to optimize qubit assignment [29]. This is useful because not all qubits are created equally and not all qubits are connected equally, so we choose the least-error-prone configuration. Finally, we used the Python package M3 (Matrix-free Measurement Mitigation, or `mthree`) to mitigate measurement-related errors [25]; it uses the quantum computer’s calibration information to further reduce errors. In the near future, these low-level operational issues are likely to be transparent to the general user. And with the development of increasingly more fault-tolerant qubits and high connectivity⁹, qubit assignment should not matter at a low level either.

For a creative application, however, noise is not necessarily bad, provided it can be somehow controlled. Manageable degrees of noise can be useful, for example, to introduce surprises in the outcomes. In the case of QuSing, we can set it to tolerate ‘wrong’ events under certain conditions. For example, it can tolerate an event that is not covered by the respective rule, provided that there exists a rule that would enable the generative process to continue from the wrong event. Otherwise, the result is discarded and the circuit is run again until it produces a permissible result.

6.2 Variability control

Also useful, is the ability to control the degree of variation of an outcome. We can do this by changing the number of previous events in the rules. The higher the number n of previous events, the higher the resemblance between the outputs and the training tune(s).

Figures 17, 18, and 19 plot the results from running QuSing for 50 generative rounds, one shot each, with rules considering $n = 1$, $n = 2$, and $n = 3$ previous events, respectively. We trained the system with an excerpt of J. S. Bach’s *Cello Suite Nr. 1* (Appendix 8.2, Figure 26) to generate music on a simulator and on quantum hardware¹⁰.

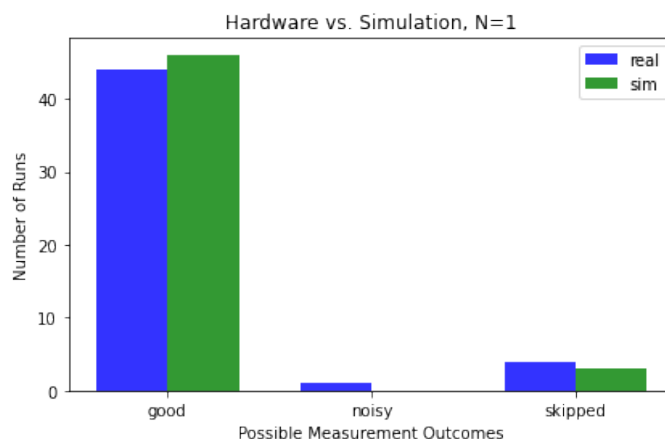


Figure 17: Results from running QuTune for 50 rounds, with rules considering $n = 1$ previous events.

⁹Gates involving more than one qubit need to be connected. But, not all qubits of a superconducting processor are connected, which limits the ability to entangle them.

¹⁰For the simulations we used IBM Quantum’s Aer simulators and for runs on real hardware we used the `ibmq_lima` and `ibmq_belem` backends.

We monitored the variability of the results by counting the number of times the system skipped building a quantum circuit because there was only one possible choice, which was retrieved classically. There were fewer choices to be made with $n = 3$ rules than with $n = 1$ ones. Therefore the results with $n = 1$ carry more variations than the results with $n = 3$. For instance in Figures 17, 18, and 19, the bars for ‘good’ count the number of times the system built circuits, whereas the bars for ‘skipped’ indicate when it has not done so. The ‘noisy’ bar indicates the number of times QuSing produced a ‘wrong’ event¹¹; obviously, simulations are unlikely to produce wrong events.

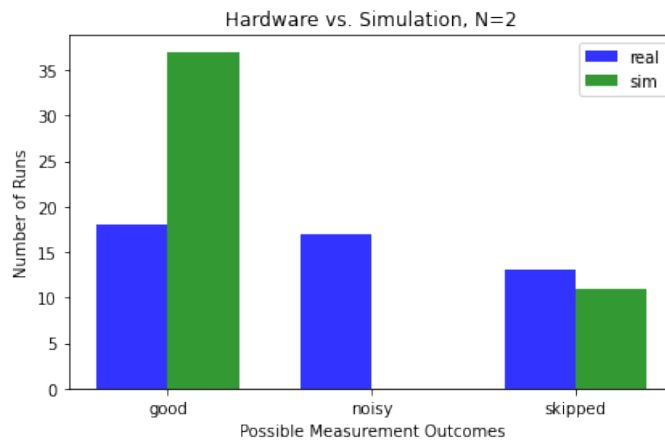


Figure 18: Results from running QuTune for 50 rounds, with rules considering $n = 2$ previous events.

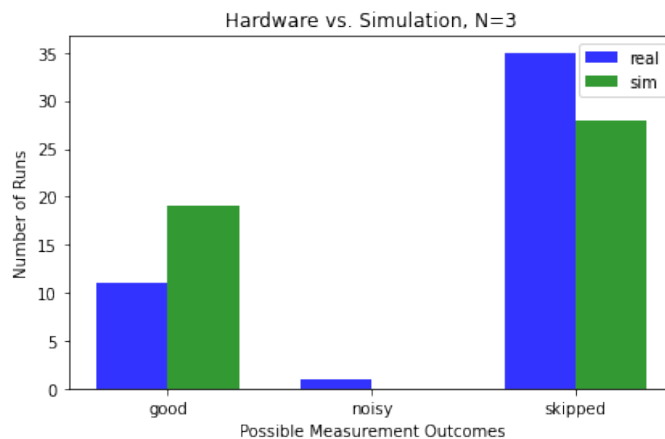


Figure 19: Results from running QuTune for 50 rounds, with rules considering $n = 3$ previous events.

Note, however, that in Figure 18 errors are prevalent. We were not able to establish the

¹¹In these examples the wrong events were not tolerated to produce the musical outputs shown in the Appendix.

reason for this. One of the reasons for this discrepancy might be that we ran this on a different backend: for Figure 18 we used the `ibmq_lima` backend, whereas for the other two we used the `ibmq_belem` one. Surely, there is a better explanation, but this is not so relevant to scrutinise at this point. Sometimes we just get a bad start: we run it and it just keeps producing wrong events. At other times, if we get a good start, then it runs smoothly. Errors increase runtime because we have to wait in a queue each time we run a circuit. We might have to wait in multiple queues to get a good result. A queue could mean we wait for a minute, but it might mean we wait for many minutes. The respective musical results are available in Appendix, section 8.2.

6.3 Saving ammunition

An error-mitigation method that is commonly applied in quantum computing is to run the same circuit many times and pick the result that occurs the most. These repetitions are referred to as *shots*.

In the case of QuSing, the number of shots can be used as a parameter to control the outcomes. For instance, consider one of the rules discussed in section 3: $D_4 \implies C_3(20\%) \vee E_3(80\%)$. Whereas running a circuit for this rule for many shots would minimise the chances of producing a wrong note (compare the ‘noise’ bars in Figures 21 and 22), it would significantly increase the chances of producing the note E_3 . That is, running a circuit for many shots unbalances the respective rule towards the option that has the highest probability, excepting, of course, those rules with equal probabilities for all possible outcomes. As an example of the effect of the number of shots, compare the tunes generated with 1 shot per round shown in Figure 16 and the one generated with 1,000 shots per round, with the same rules and backend, shown in Figure 20. In the latter, from the third musical bar onwards, the tune repeats the same motif until the end. In practice, the system got biased with only one option for most rules. This, combined with rules that already offered only one option anyway, caused the lack of variation.



Figure 20: QuSing-generated tune, with 50 rounds, 1,000 shots each.

6.4 Classic, quantum, or hybrid?

We mentioned earlier that, for rules that have only one possible outcome, the system skips the quantum processing and retrieves the only possible choice classically. For larger training inputs and with data of increased complexity, it is unlikely that there will be rules with only one outcome. On the contrary, we are likely to see rules with significantly more options with varied probabilities of occurrence. All the same, our choice of processing these cases classically is purely circumstantial, given the state of the art of technology we currently have access to. As discussed above, it would be far neater to process everything quantumly and benefit from quantum uncertainty and noise. But given the time it takes to communicate with a quantum machine over the cloud, and the fact that we currently have to do this for every new event, we might as well avoid doing this here.

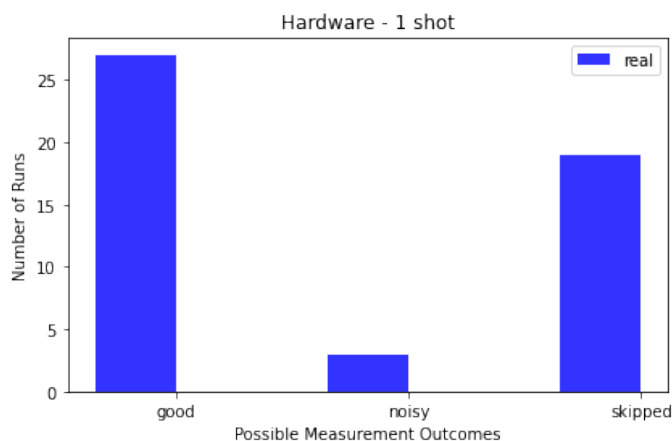


Figure 21: Results from running the circuits for 1 shot.

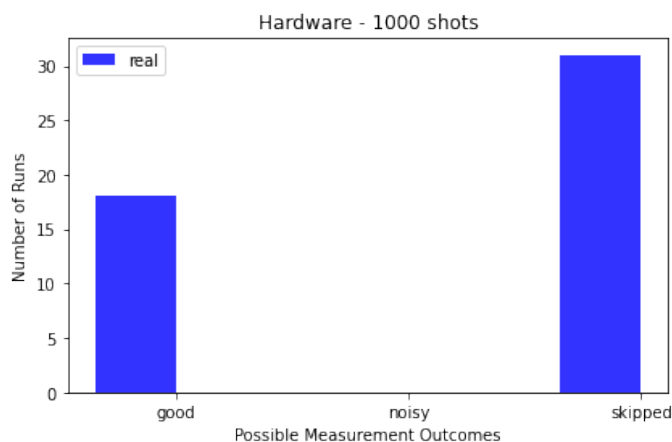


Figure 22: Results from running the circuits for 1,000 shots.

7 Final Remarks

The field of quantum computing is incipient. The current stage of quantum computing technology is somewhat comparable with the early days of classical digital computers. They also were unportable, unreliable, high-maintenance, expensive, and required highly specialized skills to be handled. It is hard to imagine what computers will be like a few decades from now, in the same way that it must have been hard for our forefathers of the 1950s to imagine how computers ended up being like today.

It is fair to say that programming a quantum computer at the time of writing is comparable to having to program a digital computer using a low-level programming language: one has to write code that operates at the level of qubits. This is changing rapidly, as the industry is making impressive progress in developing high-level software development kits (SDK); e.g., Microsoft's QDK, IBM's Qiskit, Xanadu's Strawberry Fields, and Quantinuum's TKET, to cite but four. Indeed, this paper discussed implementation and operational issues (e.g., noise, circuit optimisation, and sludgy Internet access to backends) that are likely to be transparent in the near future, but which, nevertheless, researchers developing the field should be aware of today.

At this stage, we are not in a position to advocate any quantum advantage for musical applications. What we advocate, however, is that the music technology community should be quantum-ready for when quantum computing hardware becomes more sophisticated, widely available, and possibly advantageous for creativity and business. In the process of learning and experimenting with this new technology, novel approaches, creative ideas, and innovative applications are bound to emerge. The method introduced here certainly is an example of an innovative approach to computing transition rules, which is truly *quantum native*.

By way of future work, we are in the process of increasing the amount of information that constitutes an event, which will include parameters for expressive singing. In the system presented in this paper, a musical event is constituted of two only kinds of information: pitch and duration. However, the vocal synthesisers that we are working with have several parameters to control vocal characteristics such as air inhalation and exhalation, lung pressure, mouth opening, vibrato, nasalisation, and attack time (for consonants), and more. This will require longer bit strings to represent the events, and consequently more qubits and circuits of increased complexity.

Acknowledgements

This work was developed as part of the QuTune Project¹², funded by the UK National Quantum Technologies Programme's QCS Hub. For the purpose of open access, the authors have applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising.

¹²<https://iccmr-quantum.github.io/>

8 Appendix

Audio recordings are available [38, 42].

8.1 *Mission: Impossible* Experiments

Given the *Mission: Impossible* theme shown in Figure 15 to train QuSing, Figures 23, 24, 25 shows three examples of outputs generated with $n = 1$, $n = 2$, and $n = 3$, where n is the number of previous events in the rules. These were run on IBM Quantum's Aer simulator. The system was set to produce 100 events with one shot per generative cycle.

Generative QuSing #1

$\text{♩} = 120$

6

11

15

19

23

27

31

Figure 23: Music generated with $n = 1$.

Generative QuSing #2

$\text{♩} = 120$

6

11

16

20

25

30

33

The image shows a musical score for 'Generative QuSing #2'. It consists of eight staves of music, each starting with a measure number (6, 11, 16, 20, 25, 30, 33). The tempo is marked as quarter note = 120. The music is written in a single melodic line on a treble clef staff. The notes are mostly quarter and eighth notes, with some rests and accidentals (sharps and flats). The piece ends with a double bar line at measure 33.

Figure 24: Music generated with $n = 2$.

Generative QuSing #3

$\text{♩} = 120$

The image shows a musical score for a single melodic line in treble clef. The tempo is marked as quarter note = 120. The score consists of seven staves of music, with measure numbers 6, 11, 16, 21, 26, and 31 indicated at the beginning of their respective staves. The key signature has one flat (B-flat), and the time signature is 4/4. The melody is composed of eighth and quarter notes, with various accidentals (sharps, flats, and naturals) and rests. The piece concludes with a double bar line at the end of the seventh staff.

Figure 25: Music generated with $n = 3$.

8.2 Q. C. Bach Experiments: Hardware vs. Simulations

Below are results from experiments with hardware and simulation, with $n = 1$, $n = 2$, and $n = 3$, where n is the number of previous events in the rules. The input music is an excerpt from J. S. Bach's *Cello Suite Nr. 1*, which is shown in Figure 26. The outputs are from running the system to generate 50 events. Note that the system increased the duration of the notes (semiquavers became crotchets) to make the tunes more fitting for singing. For the simulations, we used IBM Quantum's Aer simulator. For quantum hardware, we used `ibmq_lima` and `ibmq_belem` backends.

J. S. Bach: Cello Suite #1

♩ = 100

Figure 26: Original excerpt of J. S. Bach's *Cello Suite Nr. 1*.

Figures 27 and 28 shows the results from hardware and simulation, with $n = 1$.

Q. C. Bach: Hardware #1

♩ = 120

Figure 27: Quantum generated Bach, hardware, $n = 1$.

Q. C. Bach: Simulator #1



Figure 28: Quantum generated Bach, simulator, $n = 1$.

Figures 29 and 30 shows the results from hardware and simulation, with $n = 2$.

Q. C. Bach: Hardware #2



Figure 29: Quantum generated Bach, hardware, $n = 2$.

Q. C. Bach: Simulator #2



Figure 30: Quantum generated Bach, simulator, $n = 2$.

Figures 31 and 32 shows the results from hardware and simulation, with $n = 3$.

Q. C. Bach: Hardware #3



Figure 31: Quantum generated Bach, hardware, $n = 3$.

Q. C. Bach: Simulator #3

$\text{♩} = 120$

The image displays a musical score for a piece titled "Q. C. Bach: Simulator #3". The score is written in bass clef with a 4/4 time signature. The tempo is indicated as quarter note = 120. The music consists of three staves, each containing four measures. The first staff starts with a treble clef and a 4/4 time signature. The second and third staves are marked with measure numbers 5 and 10 respectively. The notes are primarily eighth and quarter notes, with some accidentals (sharps and naturals).

Figure 32: Quantum generated Bach, simulator, $n = 3$.

References

- [1] Anumanchipalli, G., Cheng, Y., Fernandez, J., Huang, X., Mao, Q., and Black A. (2010). "KLATTSTAT: Knowledge-based Parametric Speech Synthesis". *Proceedings Seventh ISCA Tutorial and Research Workshop on Speech Synthesis* Kyoto, Japan.
- [2] Arya, A., Botelho, L., Canete, F., Kapadia, D. and Salehi, O. (2022). "Music Composition Using Quantum Annealing". In E. R. Miranda (Ed.), *Quantum Computer Music: Foundations, Methods and Advanced Concepts*. Cham: Springer. Pre-print: <https://arxiv.org/abs/2201.10557> (Accessed 05 June 2022)
- [3] Bell, C. (2011). "Algorithmic music composition using dynamic Markov chains and genetic algorithms". *Journal of Computing Sciences in Colleges*, 27(2):99-107.
- [4] Bernhardt, C. (2019). *Quantum Computing for Everyone*. The MIT Press. ISBN: 978-0262039253.
- [5] Bevins, G. (2013). "Computer technology in modern music: a study of current tools and how musicians use them". *Capstone Projects and Master's Theses*, 367. California State University. Digital Commons: https://digitalcommons.csumb.edu/caps_theses/367/ (Accessed on 05 June 2022)
- [6] Bittner, R. (2022). "Meet Basic Pitch: Spotify's Open Source Audio-to-MIDI Converter". *Spotify Engineering Blog*. Online publication: <https://engineering.atspotify.com/2022/06/meet-basic-pitch/> (Accessed on 05 June 2022).
- [7] Brown, A. R., Gifford, T. and Davidson, R. (2015). "Techniques for Generative Melodies Inspired by Music Cognition". *Computer Music Journal*, 39(1):11-26.
- [8] Brown, S., Merker, B. and Wallin, N. L. (Eds.) (1999). *The Origins of Music*. The MIT Press, USA. ISBN: 978-0262731430.
- [9] Chuharski, J. M. (2022). "Adiabatic Quantum Computing and Applications to Music". In E. R. Miranda (Ed.), *Quantum Computer Music: Foundations, Methods and Advanced Concepts*. Cham: Springer.
- [10] Crosson, E. J. and Lidar, D. A. (2021). "Prospects for quantum enhancement with diabatic quantum annealing". *Nature Reviews Physics*, 3(466-489). DOI: <https://doi.org/10.1038/s42254-021-00313-6>
- [11] Dodge, C. and Jerse, T. (1997). *Computer Music: Synthesis, Composition, and Performance*. Schirmer Books, 2nd edition. ISBN:978-0028646824.
- [12] Doornbusch, P. (2004). "Computer Sound Synthesis in 1951: The Music of CSIRAC." *Computer Music Journal*, 28:1(10-25).
- [13] Grant E. K. and Humble, T. S. (2020). "Adiabatic Quantum Computing and Quantum Annealing", *Physics*, July 2020. Open access: <https://doi.org/10.1093/acrefore/9780190871994.013.32> (Accessed on 05 June 2022)
- [14] Griffiths, D. J. and Schroeter, D. F. (2018). *Introduction to Quantum Mechanics*. Cambridge University Press. ISBN: 9781107189638.
- [15] Grover, L. K. (1997). "Quantum Mechanics Helps in Searching for a Needle in a Haystack". *Physical Review Letters*, 79(2):325-328. DOI: 10.1103/PhysRevLett.79.325
- [16] Grumbling, E. and Horowitz, M (Eds.) (2019). *Quantum Computing: Progress and Prospects*. National Academies Press. ISBN: 9780309479691. DOI: <https://doi.org/10.17226/25196>

- [17] Hadimlioglu, I. A. and King, S. A. (2018). “Automated musical transitions through rule-based synthesis using musical properties”. *Entertainment Computing*, 28:59-67.
- [18] Hahn, M. (2020). “Subtractive Synthesis: Learn Synthesizer Sound Design”. *LANDR Blog*. Available online: <https://blog.landr.com/subtractive-synthesis/> (Accessed on 05 June 2022).
- [19] Hamido, O.C., Cirilo, G. A. and Giusto, E. (2020). “Quantum synth: a quantum-computing-based synthesizer”. *Proceedings of the 15th International Conference on Audio Mostly*, Graz, Austria. Open access: <https://dl.acm.org/doi/10.1145/3411109.3411135>
- [20] Hiller, L. A. and Isaacson, L. M. (1959). *Experimental Music: Composition with an Electronic Computer*. McGraw-Hill. Available online: <https://archive.org/details/experimentalmusi00hill/page/n5/mode/2up> (Accessed on 07 May 2021)
- [21] <https://www.ibm.com/quantum> (Accessed on 25 June 2022)
- [22] Kenmochi, H. and Ohshita, H. (2007). “VOCALOID - Commercial singing synthesizer based on sample concatenation”. *Proceedings of INTERSPEECH 2007, 8th Annual Conference of the International Speech Communication Association*, Antwerp, Belgium.
- [23] Kirke, A. and Miranda, E. R. (2017). “Experiments in Sound and Music Quantum Computing”. In E. R. Miranda (Ed.), *Guide to Unconventional Computing for Music*. Cham, Switzerland: Springer, pp. 121-158. ISBN: 978-3319498805.
- [24] Klatt, D. H. (1980). “Software for a cascade/parallel formant synthesizer”. *Journal of Acoustic Society of America*, 67(3):971-995.
- [25] <https://github.com/Qiskit-Partners/mthree> (Accessed on 25 June 2022)
- [26] Maestre, E., Ramirez, R., Kersten, S. and Serra, X. (2009). “Expressive Concatenative Synthesis by Reusing Samples from Real Performance Recordings”. *Computer Music Journal*, 33(4):23-42.
- [27] Manning, P. (2004). *Electronic and Computer Music*. Oxford, UK: Oxford University Press. ISBN: 978-0195170856.
- [28] Mannone, M. and Rocchesso, D. (2022). “Quanta in Sound, the Sound of Quanta: A Voice-Informed Quantum Theoretical Perspective on Sound”. In: Miranda, E.R. (Ed.), *Quantum Computing in the Arts and Humanities*. Cham, Switzerland: Springer, pp. 193-226. ISBN: 978-3030955373.
- [29] <https://github.com/Qiskit-Partners/mapomatic> (Accessed on 25 June 2022).
- [30] Mathews, M. V. and Moore, F. R. (1970). “GROOVE—a program to compose, store, and edit functions of time”. *Communications of the ACM*, 13(12):715-721. Open Access: <https://dl.acm.org/doi/10.1145/362814.362817> (Accessed on 25 June 2022)
- [31] McAdams, A. (ed.) (1987). *Music and psychology: a mutual regard. Contemporary Music Review*, Vol. 2, Part 1. Gordon and Breach, UK.
- [32] Miranda, E. R. and Basak, S. T. (2022). “Quantum Computer Music: Foundations and Initial Experiments”. In E. R. Miranda (Ed.), *Quantum Computer Music: Foundations, Methods and Advanced Concepts*. Cham: Springer. Pre-print: <https://arxiv.org/abs/2110.12408> (Accessed on 25 June 2022)

- [33] Miranda, E. R. (2021). “Quantum Computer: Hello, Music!”. In E. R. Miranda (Ed.). *Handbook of Artificial Intelligence for Music: Foundations, Advanced Approaches, and Developments for Creativity*. Springer International Publishing. ISBN: 9783030721152. Preprint: <https://arxiv.org/abs/2006.13849> (Accessed on 25 June 2022)
- [34] Miranda, E. R. (Ed.) (2021). *Handbook of Artificial Intelligence for Music: Foundations, Advanced Approaches, and Developments for Creativity*. Springer International Publishing. ISBN: 9783030721152.
- [35] Miranda, E. R. (2020). “Creative Quantum Computing: Inverse FFT, Sound Synthesis, Adaptive Sequencing and Musical Composition”. In A. Adamatzky (Ed.), *Handbook of Unconventional Computing*, pp.493-523. World Scientific. ISBN: 9789811235030. Preprint: <https://arxiv.org/abs/2005.05832> (Accessed on 25 June 2022)
- [36] Miranda, E. R. (2002). *Computer Sound Design: Synthesis techniques and programming*. Elsevier Focal Press. ISBN: 978-0240516936.
- [37] Miranda, E. R. (2001). *Composing Music with Computers*. Elsevier Focal Press. ISBN: 9780240515670.
- [38] SoundClick album: <https://www.soundclick.com/artist/default.cfm?bandID=1504503> (Accessed on 25 June 2022)
- [39] Nierhaus, G. (2009). *Algorithmic Composition: Paradigms of Automated Music Generation* Springer. ISBN: 978-3211755402.
- [40] <https://github.com/CQCL/tket> (Accessed on 25 June 2022)
- [41] https://qiskit.org/documentation/tutorials/circuits_advanced/04_transpiler_passes_and_passmanager.html (Accessed on 25 June 2022)
- [42] QuSing GitHub repository: <https://github.com/iccmr-quantum/QuSing> (Accessed on 25 June 2022)
- [43] Redman, N. (2002). “A gentle introduction to the FFT”. *Ear Level Engineering*. Online publication: <https://www.earlevel.com/main/2002/08/31/a-gentle-introduction-to-the-fft/> (Accessed on 01 July 2022)
- [44] Rieffel, E. and Polak, W. (2011). *Quantum Computing: A Gentle Introduction*. The MIT Press. ISBN: 9780262015066.
- [45] Rutledge, J. C., Cummings, K. E., Lambert, D. A. and Clements, M. A. (1995). “Synthesizing styled speech using the Klatt synthesizer”. *Proceedings of 1995 International Conference on Acoustics, Speech, and Signal Processing*. DOI: 10.1109/ICASSP.1995.479681.
- [46] Serra, X. and Smith III, J. (1990). “Spectral Modeling Synthesis: A Sound Analysis/Synthesis System Based on a Deterministic Plus Stochastic Decomposition”. *Computer Music Journal*, 14(4):12-24.
- [47] Siegelwax, B. N. (2021). *Dungeons & Qubits: An Adventurer’s Tale Beyond the Quantum Computing Tutorials*. Distributed on-line: <https://leanpub.com/dungeons-n-qubits> (Accessed on 25 June 2022)
- [48] Schwarz, D. (2005). “Current Research in Concatenative Sound Synthesis”. *Proceedings of International Computer Music Conference (ICMC 2005)*, Barcelona, Spain. Available online: <https://hal.archives-ouvertes.fr/hal-01161337/file/index.pdf> (Accessed on 05 June 2022)
- [49] Shapiro, I. and Huber, M. (2021). “Markov Chains for Computer Music Generation”. *Journal of Humanistic Mathematics*, 11(2):167-195. DOI: 10.5642/jhummath.202102.08

- [50] Strategist, Q. (2020). "What is Quantum Annealing and how does it differ from Gate based Quantum Computers?". *Quantum Zeitgeist*, October 2020. Available online: <https://quantumzeitgeist.com> (Accessed on 08 June 2022).
- [51] Villavicencio, F. and Bonada, J. (2010). "Applying voice conversion to concatenative singing-voice synthesis". *Proceedings of 11th Annual Conference of the International Speech Communication Association*, Makuhari, Chiba, Japan.
- [52] Weaver, J. (2018). "Jamming with a Quantum Computer". *Rigetti Tech Blog*. Available online: <https://www.medium.com/rigetti/jamming-with-a-quantum-computer-bed05550a0e8> (Accessed on 25 June 2022)
- [53] Wikstrom, P. (2013). "The Music Industry in an Age of Digital Distribution". *Open-Mind BBVA*. Available online: <https://www.bbvaopenmind.com/en/articles/the-music-industry-in-an-age-of-digital-distribution/> (Accessed on 25 June 2022)
- [54] Wilson, M., Chandna, P., Daido, R. and Hisaminato, Y. (2017). "Experiments in Making VOCALOID Synthesis Mode Human-like Using Deep Learning". *IPSSJ SIG Technical Reports*, Vol. 2017-MUS-114, No. 4.
- [55] Wong, T. G. (2022). *Introduction to Classical and Quantum Computing*. Rooted Grove. ISBN: 979-8985593105.