Estimating Algorithmic Information Using Quantum Computing for Genomics Applications

Sarkar, Aritra; Al-Ars, Zaid; Bertels, Koen

DOI
10.3390/app11062696

Publication date
2021

Document Version
Final published version

Published in
Applied Sciences

applied sciences

MDPI

Article

Estimating Algorithmic Information Using Quantum Computing for Genomics Applications

Aritra Sarkar [1,*], Zaid Al-Ars [1] and Koen Bertels [2]

[1] Department of Quantum & Computer Engineering, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, 2628 CD Delft, The Netherlands; z.al-ars@tudelft.nl

[2] Department of Informatics Engineering, Faculty of Engineering, University of Porto, 4200-465 Porto, Portugal; koen.bertels@qbee.eu

* Correspondence: a.sarkar-3@tudelft.nl

Abstract: Inferring algorithmic structure in data is essential for discovering causal generative models. In this research, we present a quantum computing framework using the circuit model, for estimating algorithmic information metrics. The canonical computation model of the Turing machine is restricted in time and space resources, to make the target metrics computable under realistic assumptions. The universal prior distribution for the automata is obtained as a quantum superposition, which is further conditioned to estimate the metrics. Specific cases are explored where the quantum implementation offers polynomial advantage, in contrast to the exhaustive enumeration needed in the corresponding classical case. The unstructured output data and the computational irreducibility of Turing machines make this algorithm impossible to approximate using heuristics. Thus, exploring the space of program-output relations is one of the most promising problems for demonstrating quantum supremacy using Grover search that cannot be dequantized. Experimental use cases for quantum acceleration are developed for self-replicating programs and algorithmic complexity of short strings. With quantum computing hardware rapidly attaining technological maturity, we discuss how this framework will have significant advantage for various genomics applications in meta-biology, phylogenetic tree analysis, protein-protein interaction mapping and synthetic biology. This is the first time experimental algorithmic information theory is implemented using quantum computation. Our implementation on the Qiskit quantum programming platform is copy-left and is publicly available on GitHub.

Keywords: algorithmic information theory; universal distribution; Kolmogorov complexity; quantum algorithms; quantum circuit model; quantum Turing machine; genomics; viral genomics; meta-biology

Citation: Sarkar, A.; Al-Ars, Z.; Bertels, K. Estimating Algorithmic Information Using Quantum Computing for Genomics Applications. *Appl. Sci.* **2021**, *11*, 2696. https://doi.org/10.3390/app11062696

Academic Editor: David Windridge

Received: 3 February 2021
Accepted: 12 March 2021
Published: 17 March 2021

1. Introduction

The evaluation of metrics such as the algorithmic complexity and algorithmic probability of finite sequences are key in scientific inference. For example, in biological systems, a better understanding of the algorithmic structures in DNA sequences [1] and cellular dynamics [2] would greatly advance domains such as personalized medication. Also, data-driven approaches such as deep learning and lossless compression often fail in providing insights on the causal generative mechanism underlying the set of observations about the physical process under study. In this research, we follow the algorithmic information theoretic approach of enumerating the distribution of all generating automata (e.g., Turing machines). The uncomputable nature of these metrics is approximated in practice by restricting the resources available to the computational models, such as time/cycles and space/memory.

Such approaches however remain intractable except for the simplest of cases. This is due to the exponential scaling of the number of possible automata that needs to be enumerated for experimenting on real-world dataset sizes. We explore the possibility of accelerating this technique on a (gate-based) circuit model of quantum computation. We use the design developed in [3] of the exact circuit required to simulate a superposition of resource-bounded stored-program automata. Thereafter, we present a full experimental framework for using this approach for inferring various metrics such as the algorithmic probability and algorithmic complexity. The output data strings or self-replicating programs evolve on the specified automata as quantum superposition for further downstream quantum algorithms. Although quantum search-based approaches require high qubit multiplicity and quality as compared to those available in the NISQ era, these search approaches cannot be dequantized for this use case of the unstructured and computationally irreducible database of program-output relations. That makes this particular algorithm an ideal candidate for demonstrating quantum supremacy with widespread application. We propose use cases in genomics of quantum accelerated viral genomics, meta-biology and synthetic biology.

Our approach is rooted in various automata models, computability, algorithmic information theory and resource complexity estimation techniques. The necessary background for these is presented in Section 2. In Section 3, we introduce the quantum circuit implementation for evolving a superposition of classical automata and discuss cases where conditioning the resultant universal distribution as a quantum superposition state has a computational advantage. Experimental use case for self-replicating programs and algorithmic complexity is presented in Section 4. Applications in genomics which can benefit from the developed framework are explored in Section 5. Section 6 concludes the paper.

2. Background

The background concepts for this research are presented in this section. It includes the description of a Turing machine and its various properties. The motivation for the restricted automata model used in this work is explained. Thereafter, the various algorithmic metrics are introduced.

2.1. Automata Models

The Turing machine (TM) model of computation, defines an algorithm as an initial input to final output transformation on the tape memory, by a program defined as a finite state machine. A TM manipulates symbols according to a table of transition rules on an infinite memory strip of tape divided into discrete cells. These user-specified transition rules can be expressed as a finite state machine (FSM) which can be in one of a finite number of states at any given time and can transition between states in response to external inputs. The Turing machine, as shown in Figure 1, positions its head over a cell and reads the symbol there. As per the read symbol and its present state in the table of instructions, the machine (i) writes a symbol (e.g., a character from a finite alphabet) in the cell, then (ii) either moves the tape one cell left or right, and (iii) either proceeds to a subsequent instruction or halts the computation.

A universal Turing machine (UTM) simulates an arbitrary Turing machine on an arbitrary input. It modifies the Turing machine by dropping the requirement of having a different transition table for each application. Every TM can be assigned a number, called the machine's description number, encoding the FSM as a list of transitions. A UTM reads the description of the machine to be simulated as well as the input to that machine from its own tape, as shown in Figure 2. This concept is realized as the stored-program von Neumann architecture. The existence of this direct correspondence between natural numbers and TM implies that the set of all Turing machines (or programs of a fixed size) is denumerable. However, most of these programs may not have any practical utility.

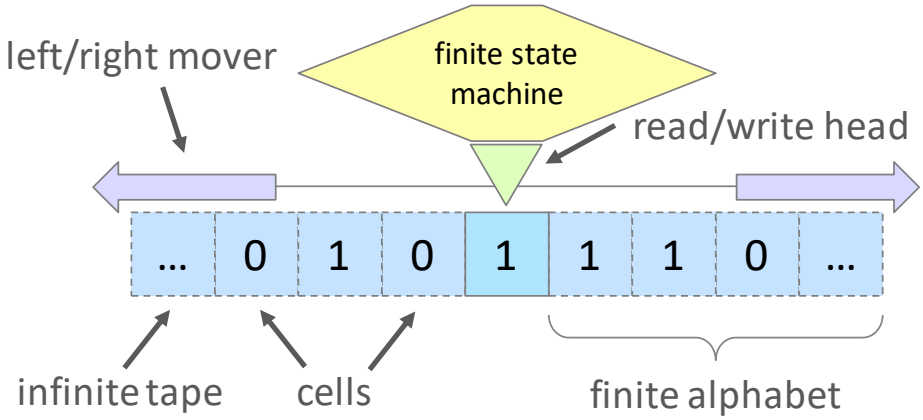


Figure 1. Computational model of a Turing machine.

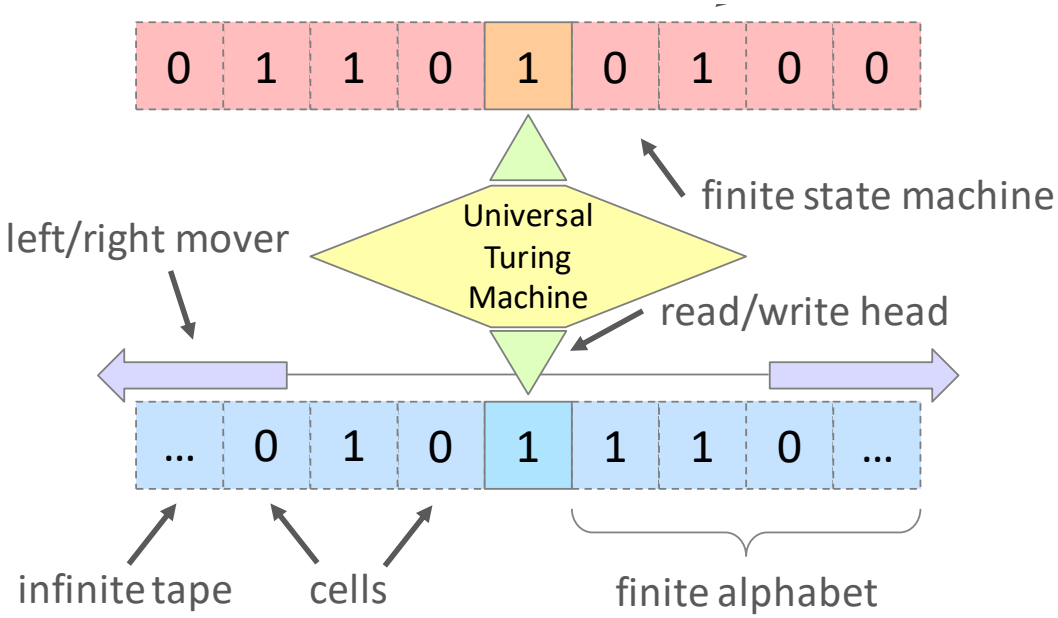


Figure 2. Computational model of a universal Turing machine.

2.2. Restricted Parallel Universal Linear Bounded Automata

While Turing machines can express arbitrary computations, their minimalist design makes them unsuitable for algorithm design in practice. Some modifications of the TM result in equivalent computation power, as captured by the Church-Turing thesis. Here, equivalence refers to being within polynomial translation overhead in time or memory resources. These models (such as lambda calculus, Post machine, cyclic-tag system) offer different perspectives in development of computer hardware and software by computing faster, using less memory, or having a smaller instruction set; however, they cannot compute more mathematical functions.

It is physically not possible for an automata to possess an infinite tape. Thus, experimental realizations of computational models restrict the tape features in various ways, by limiting the size (linear bounded automata), restricting its input-output such as a stack (push-down automata) or not having a memory altogether (finite state automata). We will consider a linear-bounded automata (LBA) with a circular tape. More specifically, we will consider a parallel universal LBA (PULBA) which allows storing the program in the memory, and multiple programs to run in parallel (by dovetailed multi-threading or in quantum superposition).

For implementing any automata, besides the memory, the runtime also needs to be restricted to a predetermined cycle count cutoff. This is because it is not generally possible to estimate the halting time of a Turing machine (or an LBA) in advance by inspecting a program's structure. Not all applications however allow a time restricted formulation. In our research, we are interested in these specific types of algorithms that can be aborted externally (with or without inspecting the current progress). The premature output yields an acceptable approximate solution with a bounded error based on the computation time. However, our assumptions on the space and time resources are motivated from the application perspectives of biological processes that we intend to model.

There is no definitive metric to compare the power of tape-restricted TMs besides assigning them to the wide class in the Chomsky hierarchy. Most landmark research [4] on the universality of TMs has assumed infinite tape length, adding a few points on the pareto curve of universality for the number of symbols and states. For restricted TMs, instead of universality, more important metrics are expressibility and reachability. Expressibility is the set of solutions/states that an automaton can compute given a non-universal set of operations but not restricted by memory/space or steps/time resources. Reachability is the set of solutions/states that an automaton can compute by restricting space and/or time resources. For our computational model of a time-bound LBA, the reachability metric will be more suitable, semantically being the set of non-zero probability strings in the universal distribution we empirically obtain.

In our model, we do not consider halting states (though it is not too difficult to include these too). Recent research [5] showed that a non-halting TM is statistically correlated in rank with LBA. Though the Chomsky hierarchy level of a non-halting LBA has not been explored, we conjecture that it will be correlated to type-1 rather than to type-2/3 in the hierarchy. The findings in [5], establish that more than 60% of algorithmic features, can be empirically tested by output distributions produced by sub-universal computation models, a key component for our research. As discussed later, restricting the TM based on the number of cycles is an unavoidable technicality as we need to collapse the quantum superposition. In the next section, we introduce the algorithmic features that we intend to estimate using our method.

2.3. Algorithmic Information

Algorithmic information theory (AIT) [6] allows studying the inherent structure of objects without reference to a generating distribution (often assumed erroneously as the uniform prior in statistical machine learning). The theory originated when Ray Solomonoff [7], Andrey Kolmogorov [8], and Gregory Chaitin [9] looked at information, probability and statistics through the algorithmic lens. The theory has now become a central part of theoretical computer science [10].

While AIT metrics are ubiquitous, most metrics are uncomputable. Estimating these using approximate methods is often intractable beyond small cases even on classical supercomputers. Thus, while in the field of theoretical computer science, these fundamental concepts are valuable for proofs, their applicability to real-world data and use cases remain very limited. The definitions and applications of some of these AIT metrics that are used in this research, are explained here.

- **Algorithmic complexity:** Algorithmic complexity (AC), also called Kolmogorov complexity, is the length of the shortest program that produces a particular output and halts on a specific TM. Formally, it is defined as:

$$AC_T(s) = \min:\{l(p), T(p) \rightarrow s\} \quad (1)$$

  where T is a TM, p is a program, $l(p)$ is the length of the program, s is a string, and $T(p) \rightarrow s$ denotes the fact that p executed on T outputs s and halts.
  AC is not a computable quantity in the general case due to fundamental limits of computations that arise from the halting problem (i.e., it is impossible to determine

whether any given program will ever halt without actually running this program, possibly for infinite time). However, it has bounded lower-approximate property, i.e., if we can find a program p_L in a language L (e.g., Java, Assembly), $l(p_L) \geq l(p)$.
Since AC depends on the particular model of computation (i.e., the TM and the language), it is always possible to design a language where a particular string s will have a short encoding no matter how random. However, the invariance theorem guarantees that, there exists a constant additive factor $c_{T1 \to T2}$ independent of s, such that

$$AC_{T2} \leq AC_{T1} + c_{T1 \to T2} \tag{2}$$

This constant is the compiler length for translating any arbitrary program p_{T1} for $T1$ to p_{T2} for $T2$.

- **Algorithmic probability:** Algorithmic probability (AP), also called Solomonoff's probability, is the chance that a randomly selected program will output s when executed on T. The probability of choosing such a program is inversely proportional to the length of the program.

$$AP_T(s) = \sum_{p:T(p) \to s} 2^{-l(p)} \tag{3}$$

Thus, the largest contribution to the term comes from the shortest program that satisfies the condition. It is uncomputable for the same reasons as AC. AP is related to AC via the following law:

$$AC_T(s) = -log_2(AP_T(s)) + c \tag{4}$$

i.e., if there are many programs that generate a dataset, then there has to be also a shorter one. The arbitrary constant is dependent on the choice of a programming language.
AP can be approximated by enumerating programs on a TM of a given type and counting how many of them produce a given output and then divide by the total number of machines that halt. When exploring machines with n symbols and m states algorithmic probability of a string s can be approximated as follows:

$$D(n,m)(s) = \frac{|T \in (n,m) : T \text{ outputs } s|}{|T \in (n,m) : T \text{ halts }|} \tag{5}$$

The coding theorem method (CTM) [11] approximates the AC as:

$$CTM(n,m)(s) = -log_2 D(n,m)(s) \tag{6}$$

Calculating CTM, although theoretically computable, is extremely expensive in terms of computation time. The space of possible Turing machines may span thousands of billions of instances.
Block decomposition method (BDM) approximates the CTM value for an arbitrarily large object by decomposing into smaller slices of appropriate sizes for which CTM values are known and aggregated back to a global estimate. The algorithmic complexity of these small slices are precomputed and make into a look-up table using CTM. The BDM aggregates this as:

$$BDM(s) = \sum_{s_i} CTM(s_i) \tag{7}$$

where $CTM(s_i)$ is the approximate algorithmic complexity of the string s_i and $s = \cup_i s_i$ i.e., the s_i together forms the string s. Small variations on the method of dividing the string into blocks becomes negligible in the limit, e.g., to take a sliding window or blocks.

- **Universal distribution:** The universal a priori probability distribution (UD) is the distribution of the algorithmic probability of all strings of a specific size. It can be calculated for all computable sequences. This mathematically formalizes the notion of Occam's razor and Epicurus' principle of multiple explanations using modern computing theory for the Bayesian prediction framework. It explains observations of the world by the smallest computer program that outputs those observations, thus, all computable theories which describe previous observations are used to calculate the probability of the next observation, with more weight put on the shorter computable theories. This is known as Solomonoff's theory of inductive inference.
- **Speed prior:** The universal distribution does not take into account the computing resource (tape and time) required when assigning the probability of certain data. This does not match our intuitive notion of simplicity (Occam's razor). Jürgen Schmidhuber proposed [12] the measure speed prior (SP), derived from the fastest way of computing data.

$$SP_T(s) = \sum_{z=1}^{\infty} 2^{-z} \sum_{p:T(p,z)\to s} 2^{-l(p)} \tag{8}$$

  where program p generates an output with prefix s after $2^{z-l(p)}$ instructions. Thus, it is even more difficult to estimate it as all possible runtimes for all possible programs need to be taken into account.
- **Omega number:** The Omega number, also called Chaitin constant or halting probability of a prefix-free TM T is a real number that represents the probability that a randomly constructed program will halt.

$$\Omega_T = \sum_{p} 2^{-|p|} \tag{9}$$

While many of the algorithmic metrics discussed here are uncomputable, most of them are either upper or lower semi-computable, i.e., the value of the metric can be approached in the limit from above or below by enumerating the Turing machines [13]. These approximations can further be calculated by time-bounding the TMs for a maximum number of cycles. Though time-bounded algorithmic metrics were researched before for various applications, pragmatically they remain intractable for real-world problem sizes for classical algorithms, motivating this research to explore alternative quantum approaches.

It is crucial to highlight that most algorithmic metrics are higher in the computing hierarchy than the more familiar polynomial-time (P) and non-deterministic polynomial time (NP) complexity classes. Since quantum computers are not expected to provide exponential speedup for NP-hard problems, we expect at best a quantum (Grover) search type polynomial speedup. In this research, we explore the possible advantages for some specific cases of the quantum model and their associated applications.

Often polynomial speedup between different computing models of the same power are heavily influenced by the formulation of a problem and the model itself. For example, it is highly non-trivial to do a binary search on a Turing machine without a random-access-memory. In this work we ensure that the quantum algorithm formulation retains the speedup. Secondly, for real-world data, often a polynomial speedup is enough in making an application tractable. In the era of small-scale quantum computers, it is thus important to explore polynomial speedups as well and better understand average case complexity and other linear and constant factors in the complexity. Thus, an experimental approach using available small-scale quantum processors or quantum simulators aids in appreciating the exact cost involved in implementing the quantum algorithm in the circuit model. In the field of quantum computing we found very few results for algorithmic metrics. Recent research [14] on reinforcement learning shares a similar motivation; however does not provide a detailed gate (time) and qubit (space) level analysis or implementation.

3. Computation Model

In our computation model, we will restrict a m states, n symbols, d dimension tape LBA by limiting the maximum time steps t before a forced halt is imposed. This automatically bounds the causal cone on the tape to $[-t, +t]$ from the initial position of the tape head. The tape is initialized to the blank character as this does not reduce the computational power. This can be thought of as, the initial part of the program prepares the input on the tape and then computes on it. The tape length, like the RAM size of a computer, is an application specific hyperparameter chosen such that it is enough for accommodating the intermediate work-memory scratchpad and the final output. The range of values for the tape length is $c \leq (2t + 1)$.

3.1. Quantum Implementation

The detailed design of the quantum circuit to implement the computation model is presented in [3]. Here, we present a summary of our implementation, which has the uniqueness in:

1. presenting a mechanistic perspective where the quantum circuit has the corresponding functions of a classical universal Turing machine,
2. this allows the user to readily translate a superposition of classical programs for a Turing machine (e.g., FSMs from assembly language code) as input states, in contrast to the cellular automata-based construction in [15],
3. Turing machine's mechanistic model does not preserve locality and homogeneity, thus reducing both the number of qubits and gate operations required to execute the automata compared to [15]
4. the core value of our construction stems from the feature that, in our model, the program can also be in a superposition along with the input data, thus allowing a superposition of classical functions to be evolved in parallel (this feature is denoted by the 'P' in QPULBA) and is generally not true for a description of a QTM.
5. In [3] We provide a complete and scalable circuit description of the full construction in two popular quantum programming languages and provide simulation results taking into account realistic resource assumptions on the runtime and qubits
6. thus, besides being a theoretical computation model, our implementation has practical applicability in the field of experimental algorithmic information theory, where the space of program-output behaviors needs to be explored exhaustively in a classical supercomputer. Thus, it forms the framework the application presented in this article.

Thus, the QPULBA acronym expansion of 'quantum', 'parallel', 'universal', 'linear bounded' translates respectively to the automata features of a superposition in inputs, a superposition of programs, a stored-program model and a memory restricted implementation. The t cycle and tape length c restricted version of QPULBA is termed QPULBA^{tc}. This is illustrated in Figure 3.

Our implementation provides a scalable quantum circuit implementation of the QPULBA model based on the 5 parameters: $\{m, n, d, t, c\}$. The copyleft implementation on the Qiskit quantum programming language can be found at https://github.com/Advanced-Research-Centre/QPULBA (accessed on 3 February 2021). The blocks of the quantum circuit for a single QPULBA step is shown in Figure 4. The blocks needs to be repeated for t cycles. The unitary blocks are purposefully named corresponding to the functions of the unitary to the classical Turing machine (read, fsm, write, move). We note that this however, does not imply the qubits are copied violating the no-cloning principle, such as in the read unitary, but are rather entangled. In our detailed circuit design, as available in [3], these unitary blocks are translated to standard quantum logic using the universal gateset (Hadamard, X, CNOT, Toffoli), implemented using a quantum programming language and tested on a quantum computing simulator. The detailed quantum circuit for the 2 state 2 symbol case is presented in Appendix A for reference. In this paper, we focus on using the QPULBA quantum circuit for various applications.

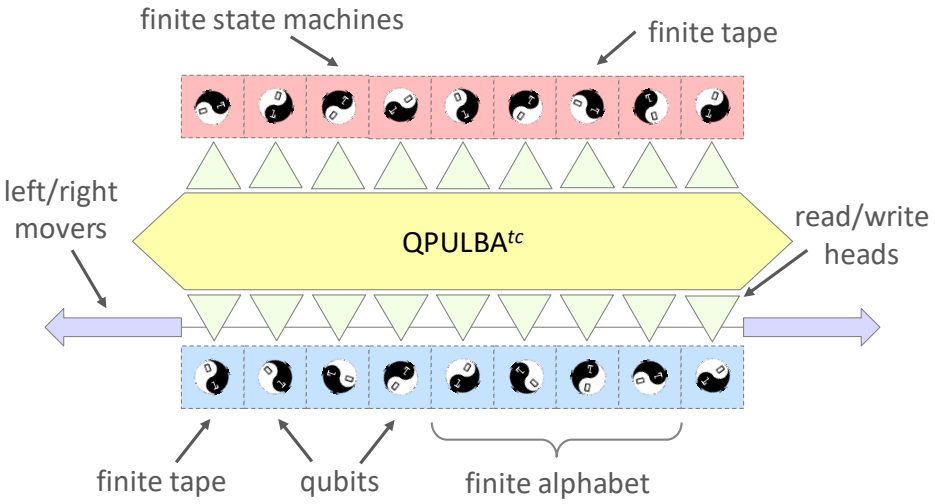


Figure 3. QPULBA^{tc}: Quantum Parallel Universal Linear Bounded Automata restricted by t cycles and tape length c.

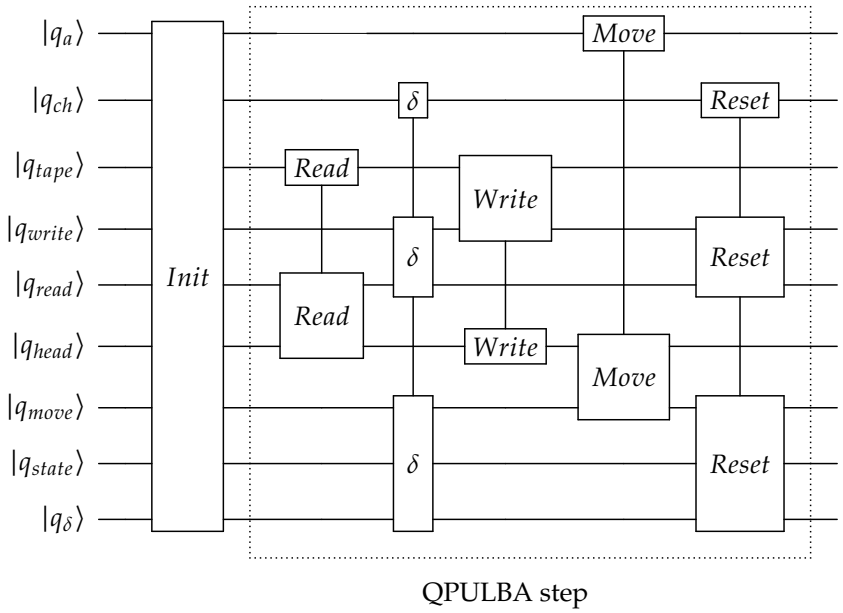


Figure 4. Blocks for the quantum circuit implementation of a QPULBA step.

3.2. An Enumerated Example

An example of this computation model is enumerated here. These enumerations are intended to be executed in parallel thereby presenting the PULBA variant of the ULBA model. For this example, we chose $c = t$ and a circular tape. This allows us to compare the final tape and program to infer which programs can self-replicate, which motivates our future research in functional genomics. The circular tape gives us similar final tape diversity (sometimes in a flipped manner) while reducing the tape resource (qubits). This shorter tape comes at the cost of not able to capture cases where part of the tape is unaffected from its initial state, which is not so interesting.

The 2-state 2-symbol 1-dimension case is both non-trivial as well as within the bounds of our current experimentation. The number of states $m = 2$ with the state set $Q : \{Q_0, Q_1\}$. The alphabet is $\Gamma : \{0, 1\}$, thus, $n = 2$ (the binary alphabet). This gives the number of qubits required for the transition function: $q_\delta = (m * n) * (log_2(m) + log_2(n) + d) = 2 * 2 * (1 + 1 + 1) = 12$ and the number of programs: $P = 2^{q_\delta} = 2^{12} = 4096$.

The machine is run for $t = q_\delta = 12$ iteration. The initial tape of is all-zeros of length $c = t = 12$. The program (description number) is encoded as: $[QMW]^{Q_1R_1}[QMW]^{Q_1R_0}$ $[QMW]^{Q_0R_1}[QMW]^{Q_0R_0}$. A Python script (the classical kernel we intent to accelerate) emulates our restricted model of the Turing machine for all 4096 cases. The program is available in the following link: https://github.com/Advanced-Research-Centre/QuBio/blob/master/Project_01/classical/ (accessed on 3 February 2021).

The tape output for all the 4096 machines is given in Figure 5 while the universal distribution is listed in Figure 6.

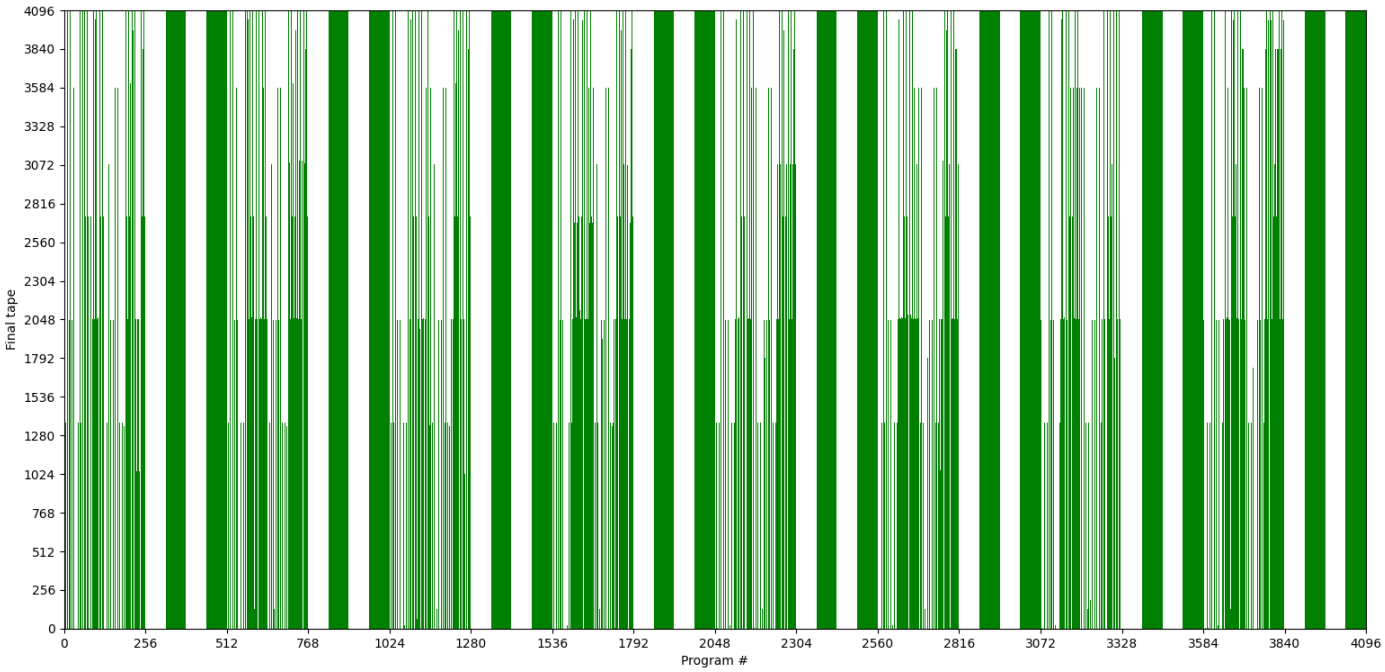


Figure 5. QPULBA(m, n, d, t, c) universal distribution for Case $m = 2$, $n = 2$, $d = 1$, $t = 12$, $c = 12$, showing the tape output for the entire space of programs.

0000 : 0.420410	1024 : 0.002197	2045 : 0.001953	2753 : 0.000244	3456 : 0.000244
0001 : 0.002197	1027 : 0.000488	2047 : 0.031250	2944 : 0.000244	3583 : 0.017578
0002 : 0.001953	1028 : 0.000244	2048 : 0.043945	3057 : 0.000244	3584 : 0.000244
0003 : 0.000244	1040 : 0.000488	2049 : 0.001465	3072 : 0.001465	3585 : 0.007080
0004 : 0.000244	1045 : 0.000244	2050 : 0.000732	3073 : 0.001465	3587 : 0.000977
0007 : 0.000732	1048 : 0.000244	2051 : 0.000244	3074 : 0.000244	3589 : 0.000244
0008 : 0.003174	1055 : 0.000244	2055 : 0.004883	3075 : 0.007080	3615 : 0.002197
0015 : 0.000488	1282 : 0.000244	2058 : 0.002441	3077 : 0.000244	3713 : 0.000244
0021 : 0.002441	1344 : 0.002441	2061 : 0.000244	3079 : 0.001221	3840 : 0.004883
0024 : 0.000244	1345 : 0.000244	2062 : 0.000244	3083 : 0.000244	3841 : 0.001221
0027 : 0.000244	1365 : 0.031250	2063 : 0.002686	3087 : 0.000244	3968 : 0.002686
0065 : 0.000488	1535 : 0.001953	2079 : 0.002441	3088 : 0.000244	3969 : 0.000244
0128 : 0.003174	1536 : 0.000244	2113 : 0.000244	3098 : 0.000244	4032 : 0.002441
0192 : 0.000244	1537 : 0.000488	2175 : 0.000244	3103 : 0.000732	4033 : 0.000732
0193 : 0.000244	1539 : 0.000488	2560 : 0.000732	3198 : 0.000244	4035 : 0.002197
0256 : 0.000244	1728 : 0.000244	2561 : 0.000244	3199 : 0.000488	4080 : 0.000244
0257 : 0.000244	1792 : 0.000732	2565 : 0.000244	3329 : 0.000244	4081 : 0.000488
0512 : 0.001953	1920 : 0.000488	2688 : 0.002441	3330 : 0.000244	4093 : 0.017578
0517 : 0.000244	1985 : 0.000244	2730 : 0.031250	3331 : 0.000244	4095 : 0.312500

Figure 6. QPULBA(m, n, d, t, c) algorithmic probability (in red) for Case $m = 2$, $n = 2$, $d = 1$, $t = 12$, $c = 12$ (for non-zero probabilities).

3.3. Quantum Advantage

Our quantum implementation of the QPULBA generates the unitary U in the standard circuit formulation in OpenQASM for Qiskit, based on the provided parameters. This is shown in Figure 7. In this section, we will present the method to use U to estimate algorithmic metrics.

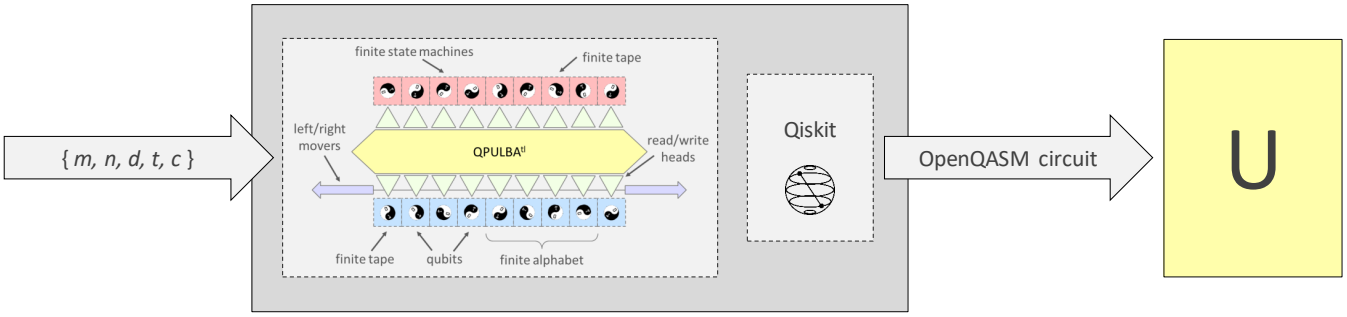


Figure 7. QPULBA circuit implementation, as detailed in [3].

The unitary U typically takes in a set of qubits representing the description number (alternatively, programs, transition table, or finite state machines). For estimating algorithmic metrics, we want to enumerate over all possible programs, thus, these qubits are put into an equal superposition as part of the initialization process. The tape qubit register is initialized to the all-zero (blank) state. The initial state qubit register is initialized to state 0 of the automata. The other qubits required for the QPULBA mechanism are not required for further discussion. These include, read, write, move, tape head, computation history and ancilla qubits.

Five cases of applying U for experimental algorithmic information theory (EAIT) are discussed here. The Wolfram Physics Project [16] correlates causal invariance with the action principle in physics and algorithmic probability in information theory. It stresses that knowing the rules (programs) of an automata does not mean it is possible to jump ahead in time without equal amount of computation. This is called computational irreducibility. However, there are some pockets of reducible computation which allows us to predict the mechanics based on the laws of physics. Akin to this concept, the quantum advantage of using QPULBA for EAIT cannot be in the time complexity of a single execution as it takes the same scaling of resources of quantum and classical gates to run a automata for a specified number of cycles. The quantum advantage must be extracted by losing information about the full quantum state vector such that some global statistical property is extracted efficiently via the quantum method. This provides an advantage with respect to classical exhaustive enumeration. Thus, the space of program-output relations is one of the best candidate problem for demonstrating speedup using quantum search.

3.3.1. Reconstructing the Universal Distribution

The universal distribution is obtained as a quantum superposition of the tape qubits at the end of the computation. However, it needs to be sampled by measurement. To reconstruct the distribution of Figure 5, the experiment needs to be repeated at least the same number of times as the number of data points. Thus, in general, there can be no quantum advantage if we aim to construct the universal distribution in full resolution.

3.3.2. Finding Output with Highest Algorithmic Probability

The mapping between programs and output can be represented as a bi-partite directed graph with a many-to-one relation. The output with highest algorithmic probability is thus the node in the output set of nodes with the largest in-degree. This node can be estimated as the statistical modal value on sampling the superposition of output. This is equivalent to reconstructing a sub-sampled approximation of the universal distribution. Similar

approximation (in terms of statistical distance such as KL divergence) can be achieved by sampling from the initial superposition of program and enumerating them classically. In terms of the graph theoretical perspective, the degree distribution of an Erdős–Rényi random graph stays similar with change in the edge probability. Thus, in this case, there is no quantum advantage as well.

3.3.3. Finding Specific Program-Output Characteristics

The quantum implementation is useful when we intend to understand a specific property of the space of programs or outputs. Such metrics are not possible to infer in classical computing without enumerating every possible machine. In the QPULBA model, the final tape distribution can be modified further based on the required application.

For example, if we intend to investigate the space of machines that self-replicate, i.e., the tape output is same as the corresponding program, the tape qubits can be evolved to the Hamming distance with respect to the FSM (program) qubits using the CNOT gate [17]. Thereafter, the zero Hamming distance can be sampled to evolve the entangled FSM qubits to a superposition of only self-replicating programs. Since the universal distribution is unstructured, a classical heuristic approach that encodes a quantum superposition of self-replicating programs is not possible without exhaustively enumerating all automata configurations. We will present an implementation of this case in Section 4.

3.3.4. Finding Algorithmic Probability of a Specific Output

Another useful application of the above case is to simply count the number of cases which generate a specific output using the quantum counting algorithm. This is equivalent to finding the algorithmic probability of a specific output. For this case, the output qubits can be evolved to the Hamming distance with respect to the required output with a series of X gates. Thereafter, we can increasingly approximate the algorithmic probability based on the measurement statistics of the zero Hamming distance state. Alternatively, we can use sampling to evolve the state to a distribution of programs that generates the specific output for further downstream quantum algorithm for analysis.

3.3.5. Finding Programs with Specific End State

The qubit register storing the final state after running the QPULBA circuit can be used to condition the universal distribution. For example, this framework can be used to count the number of cases that reach a particular state (e.g., a state denoted as the halting state, or an accepting state for the automata). Thus, it is possible to estimate the Omega number, or the halting probability for this limited case of a runtime restricted LBA model. The results from finding the probability of a specific output and the probability of the programs reaching a specific end state can be used together to estimate the Kolmogorov complexity using the coding theorem method.

4. Experimental Use Cases

We explore two use cases of quantum-accelerated experimental algorithmic information theory (QEAIT) in this section. One is to find self-replicating programs, and the second is to estimate the algorithmic (Kolmogorov) complexity using the block decomposition method.

4.1. Distribution of Quines

Universal constructor is a self-replicating machine foundational in automata theory, complex systems and artificial life. John von Neumann was motivated to study abstract machines which are complex enough such that they could grow or evolve like biological organisms. The simplest such machine, when executed, should at least replicate itself.

As shown in Figure 8, the design of a self-replicating machine consists of:

- a program or *description* of itself

- a *universal constructor* mechanism that can read any description and construct the machine or description encoded in that description
- a *universal copier* machine that can make copies of any description (if this allows mutating the description it is possible to evolve to a higher complexity)

Additionally, the machine might have an overall *operating system* (which can be part of the world rule or compiler) and *extra functions* as payloads. The payload can be very complex like a learning agent, such as AIXI [18] or an instance of an evolving neural network [19].

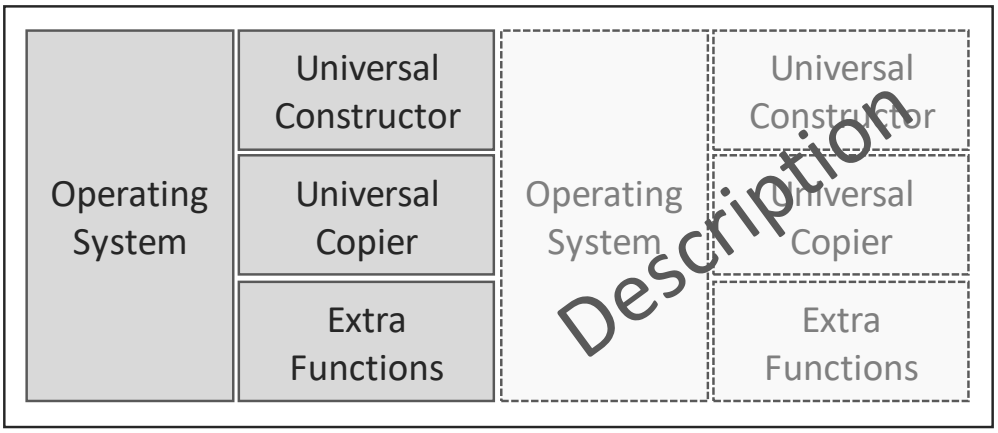


Figure 8. John von Neumann's system of self-replicating automata.

The constructor mechanism has two steps: first the universal constructor is used to construct a new machine encoded in the description (thereby interpreting the description as program), then the universal copier is used to create a copy of that description in the new machine (thereby interpreting the description as data). This is analogous to the cellular processes of DNA translation and DNA replication, respectively. The cell's dynamics is the operating system which also performs the metabolism as the extra functions when it is not reproducing.

A quine is a program which takes no input and produces a copy of its own source code as its output. Thus, it is akin to the software embodiment of constructors. Quine may not have other useful outputs. In computability theory, such self-replicating (self-reproducing or self-copying) programs are fixed points of an execution environment, as a function transforming programs into their outputs. Quines are also a limiting case of algorithmic randomness as their length is same as their output.

The idea of using the fixed-point, called the Y-combinator in lambda calculus $\lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$ to describe the genetic code [20] is pioneered by Gregory Chaitin [1] as the field meta-biology. This is in line with the constructor theory [21] approach for understanding physical transformations in biology. In the field of transcendental/recreational programming, the DNA structure was used to code the Gödel number (similar to the description number) of any Ruby script [22]. In our future research [23], we intend to explore the significance of these results for artificial life applications [24] in synthetic biology. The space and probability of constructors [25] can inform the subset of DNA encoding for in vitro experimentation and understanding of causal mechanisms in cells [2,26].

Implementation Using the QEAIT Framework

As an experimental demonstration of this use case we intend to create an equal superposition of quines for a specific automata. In the corresponding classical case, every description number encoding needs to be enumerated. Then, each output needs to be

evaluated to find self-replicating behavior. The quines are then selected and put into a quantum superposition by encoding the states in equal superposition.

Our implementation of QPULBA is scalable to any m-state n-symbol QPULBA. The entire circuit for the 1-state 2-symbol case requires much less qubits, thus we were able to simulate it classically. Please note that there is no need to store the state anymore (only 1 state) thereby reducing the qubit complexity greatly.

The full circuit was simulated for four cycles on Qiskit as discussed in Section 3.1. The code can be found at https://github.com/Advanced-Research-Centre/QPULBA (accessed on 3 February 2021). The final state vector obtained after four cycles is shown on the left side of Figure 9. The FSM qubits encoding the description/program number (in green) and the output on the tape (in red) bit strings is the universal distribution for this automata. Thus, if we measure only the tape in the standard computational basis, we will obtain an equal statistical distribution of the 0000 and 1111 states. The tape is then evolved to the Hamming distance with respect to the FSM. This results in the quantum state as shown on the right side of Figure 9.

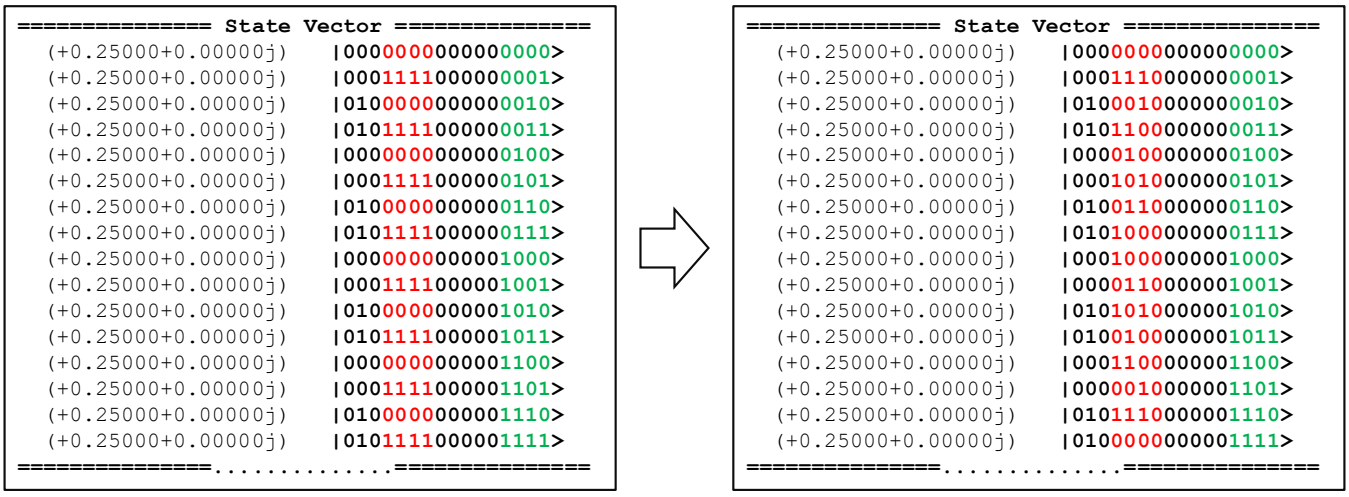


Figure 9. QPULBA 1-2-1 showing the FSM description/program number (**green**) and output tape (**red**): on left the universal distribution and on right the Hamming distance between tape and FSM.

The 0000 tape state is then marked on the MSQ (in blue), as shown on the left size of Figure 10. On sampling the 1 state of this qubit evolves the quantum state to an equal distribution of quines.

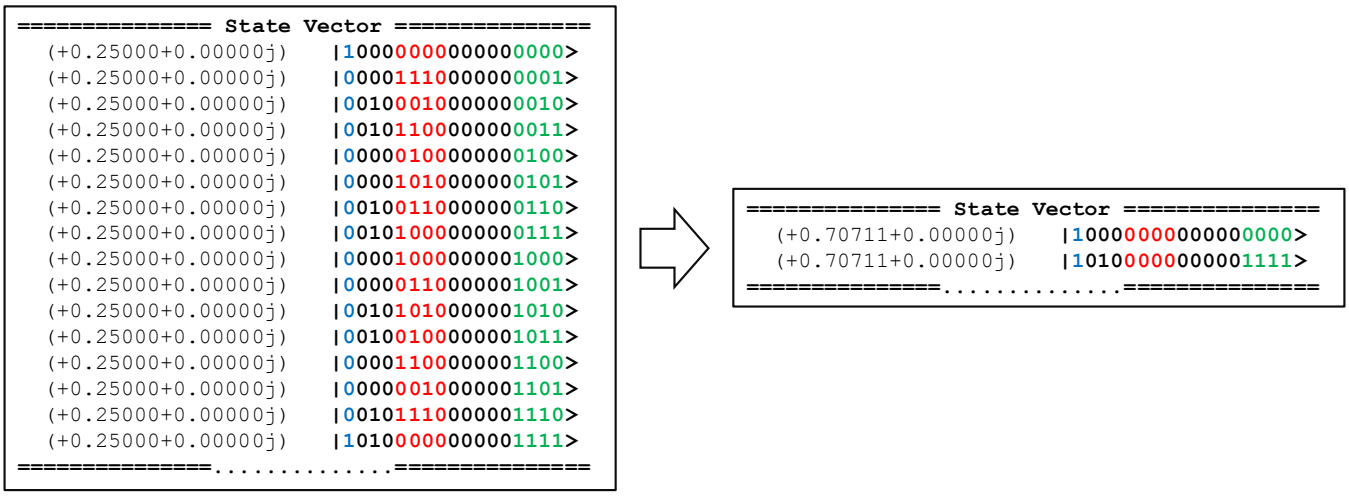


Figure 10. Sampled distribution of quines.

If the total number of quines to the total number of programs is $\frac{p_q}{p}$ and the time to run the automata is t, the classical approach takes time in the order of $O(t * (p_q + p))$, as, first we need to run all the programs to evaluate its self-replicating behavior and then create a quantum superposition of the quines. In contrast, since all the programs are executed in superposition, the quantum sampling approach takes time proportional to $O(t * (p/p_q))$.

4.2. Estimation of Algorithmic Complexity

In [13], the coding theorem method is used to estimate the algorithmic complexity of short strings. This forms the classical kernel that we accelerate with our quantum formulation. Estimation of the algorithmic complexity of sequences is actively researched by Hector Zenil and been applied to diverse fields such as evaluations of structures in genomics [27], psychometrics [28], cellular automata [29,30], graph theory, economic time series [31,32], and complex networks [28]. It is thus promising to develop a quantum computing approach, as any benefit will be magnified by the wide applicability of this technique.

In this use case, we explore the possibility to accelerate the estimation of algorithmic (Kolmogorov) complexity. As defined formally in Equation (1), this is an absolute measure of the information content of a string. A seminal result in estimating AC [33] via Experimental Algorithmic Information Theory (EAIT)-based approach considers the property that many of the algorithmic metrics are semi-computable and converges in the limit. This is crucial, as otherwise, any hope of approximating these quantities with computing models would not be possible. An important distinction can be made with 'quantum Kolmogorov complexity' which deals with the metrics to define the complexity of a specific quantum state and its preparation process using quantum gate. Our focus in this research is on 'quantum computing approaches for estimation of the algorithmic complexity of classical sequences'.

Estimating the complexity of real-world data needs a scalable approach for long data streams or graphs. The complexity of long strings can be estimated using the block decomposition method as defined in Equation (7). The estimation of complexity by BDM is compatible with lossless compression algorithms but can go beyond the scope of Shannon entropy-based methods, which are widely used to estimate AC. Entropy=based techniques can capture simple statistical patterns (repetitions), while algorithmic patterns (such as 12345... or 246810...), are characterized as having maximum randomness and the highest degree of incompressibility by these methods. These methods are not invariant to language choice and are therefore not robust enough to measure complexity or randomness as for AIT-based methods. In contrast, techniques based on algorithmic metrics can detect causal gaps in entropy-based methods (e.g., Shannon entropy or compression algorithms). BDM is a smooth transition between the Kolmogorov entropy and Shannon entropy, depends respective on whether the size of s_i is same as s or 1. Calculating the CTM value gets exponentially difficult with string size, thus the BDM is the method of choice for using algorithmic metrics for causal inferences. However, for longer data such as real-world use cases, BDM starts getting less effective (for a fixed block size) and fails to find causal links beyond the block size. To maintain the advantage over the simpler data-driven approaches we need to have an application motivated block size, e.g., how far in a genome can one gene affect the other. This motivates our research in using quantum-acceleration to extend the lookup table of the Algorithmic Complexity of Short Strings (ACSS) [34] built using the CTM technique. The quantum technique is not to supplant the BDM but to power it with the resources from the quantum-accelerated CTM.

The ACSS database is constructed by approximating the output frequency distribution for Turing machines with 5 states and 2 symbols generating the algorithmic complexity of strings of size $\leq$ 12 over $\approx$ 500 iterations. These numbers of 5, 2, 12 and 500 are arbitrary from the BDM application perspective and fixed based on the computational resource. Quoting the paper "The calculation presented herein will remain the best possible estimation for a measure of a similar nature with the technology available to date, as

an exponential increase of computing resources will improve the length and number of strings produced only linearly if the same standard formalism of Turing machines used is followed." This immediately translates to the promise of mapping this exponentially growing states using quantum computation for the Turing machine enumeration. The symbol set of the binary alphabet is justified as the standard used in data encoding in current digital computation due to its simplicity. The argument for 5 state is more directly related to computability restrictions. Being uncomputable, every Turing machine in-principle needs to be run forever to know if it will halt. However, for small Turing machines, it is possible to enumerate and classify every Turing machine as either it halts with a maximum time step of B, called the busy beaver runtime, or goes on forever. This value can be used to stop the computation and declare the enumeration as non-halting if the halt state is not reached in B steps. The value of B is known to be 107 for Turing machines with 4 states and 2 symbols. However, for 5 states the value is still unknown as there are too many cases to enumerate (26559922791424) and from partial enumerations so far, we know that the value is $\geq 47,176,870$ steps. It is intractable to run so many machines for so long iterations. AC can be estimated using a far lower number of iterations when the B value is unknown, as the number of halting TM decay exponentially [35] with steps using the random variable model, $P(S = k|S \leq S_{lim}) = \alpha e^{-\lambda k}$. For the 5 state, 2 symbol case with $S_{lim} = 500$ it is 6×10^{-173}, thus can be safely ignored. The run limit should capture on the output tape almost all 12 bit strings thus allowing the calculation of their CTM values. For 13 bits strings, only half of all possible strings were captured in 500 steps, setting the block size limit using this method. The classical technique used various intelligent optimizations such as symmetries, look ahead and patterns, reducing the requirement to only 4/11 of all machines to be enumerated for 500 steps (if they did not halt before). This however still took 18 days on a supercomputer at the Centro Informático Científico de Andalucía, Spain [13].

Analysis Using the QEAIT Framework

Estimating the algorithmic probability using the CTM requires counting the number of Turing machines that generate a particular output and evolves to a specific state. This requires at the least a 2 symbol 2 state automata to differentiate between fixed-length strings, and the halting state, respectively. Our estimates for the QPULBA-2-2-1 requires over 54 qubits, constraining us from demonstrating the full pipeline on a quantum computing simulator. However, the number of logical qubits required is promisingly low compared to other representative quantum advantage pursuits, for example in cryptography and optimization. We discuss some theoretical results in this application to consolidate our use case.

$\#P$ is the computational complexity class of all problems which can be computed by counting Turing machines of polynomial time complexity [36]. Toda's theorem states the entire polynomial hierarchy PH is contained in P^{PP}. Since $P^{PP} = P^{\#P}$ [37] as a corollary, the entire polynomial hierarchy PH is also contained in $P^{\#P}$. This signifies, if there is an efficient classical algorithm which could perform exact counting, then the polynomial hierarchy would collapse. Since we suspect this is not the case, it is unlikely that there exists a classical algorithm which can compute exact counting.

Approximate counting can be done probabilistically with a polynomial runtime (with respect to the error and string size) with an *NP-complete* oracle. Thus, if a quantum algorithm could perform approximate counting in polynomial time, then that would imply that $NP \subseteq BQP$, which is implausible. However, quantum computing might provide a polynomial speedup in solving counting problems with respect to classical computers. The quantum counting algorithm, boson sampling and post-selection are three ways that hold promise in this direction. We discuss the later two possibilities before evaluating the quantum counting approach.

Postselection is the process of ignoring all outcomes of a computation in which an event did not occur, selecting specific outcomes after (post) the computation. However,

postselection is not considered to be a feature that a realistic computer (classical or quantum) would possess, but nevertheless are interesting to study from a theoretical perspective. The complexity class $PostBQP$, is the class BQP with postselection. It was shown [38] that $BQP \subseteq PostBQP = PP$, and as a corollary $PH \subseteq P^{PostBQP}$, as $P^{PostBQP} = P^{\#P}$. This means that if there was an efficient way to postselect with a quantum computer, we would be able to solve many of the problems which are intractable classically, including exact counting. However, it is not clear how to practically design such an algorithm.

Boson sampling [39] is a proposed (non-universal) model of quantum computation which involves sampling from a probability distribution of non-interacting bosons. Sampling from bosonic or fermionic distributions can be done in polynomial time [40] using a universal quantum computer. For this it is required to calculate the permanent of a matrix encoding the state probabilities of the bosonic system. Calculating the permanent of a matrix, or even approximating it, is a *#P-hard* problem. The existence of a classical algorithm which could efficiently compute exact (or approximate) boson sampling would imply that the polynomial hierarchy collapses to the third level, which is unlikely. Thus, large boson sampling computers would allow us to solve hard problems such as the exact or approximate counting. Recently, quantum supremacy [41] was demonstrated using boson sampling; however, feasibility of practical applications are yet to be explored.

The quantum counting approach is more malleable for concrete circuit design for gate-based quantum computing. This was proposed [14] for accelerating the speed prior, another important AIT metric. Efficient approximations of Solomonoff prior would provide a powerful form of compression and prediction, for example from the artificial general intelligence agent perspective. The speed prior, as defined in Equation (8), accounts for the running time of each program on a UTM, in contrast to running each program until it halts, as in the Solomonoff prior. The speed prior, S, is an approximation, yet takes time scaling exponentially in the length of the largest program. Computing S essentially involves counting the number of programs of given length that runs in polynomial time. Thus, this is an NP problem, and the counting is $\#P$. It was conjectured that S is *#P-hard*, such that if there did exist a quantum algorithm which could solve S in polynomial time, then the polynomial hierarchy would collapse. However, a fixed length speed prior can yield a quadratic speedup using quantum computing compared to the classical method.

The speedup of both the CTM and speed prior depends on the quantum counting algorithm. Here, we present the algorithm for the quantum-accelerated CTM.

Algorithm 1: Quantum counting CTM algorithm

```
Given string s;
for i < t do
    Run QPULBA step;
end
num_s := QCount(tape = s);
for i < t do
    Run QPULBA step;
end
num_h := QCount(state = halt);
```

Result: $CTM(s) := -log_2(num_s/num_h)$

Please note that we need to run the superposition of automata twice, since the quantum state collapses on measurement. The QPULBA is run once to estimate the number of machines that output the required string, and again to count the number of halting states. The count subroutine $QCount$ is what entails the quadratic speedup. The quantum counting algorithm [42] is a combination of Grover search and phase estimation. Given an oracle indicator function f_B over a set of size $N = 2^n$, the quantum algorithm estimates $M = |B|$ by solving for θ in $\sin^2\left(\frac{\theta}{2}\right) = \frac{M}{2N}$ Recently, three new approaches to quantum counting without quantum Fourier transform were published [43–45]. Implementing these on the

Qiskit framework and integrating with the CTM algorithm will be explored in our future work.

5. Application Framework

The model of computation discussed in this research can estimate time-bound algorithmic metrics. Here, we present a framework for empirical experimentation on a quantum accelerator. This is shown in Figure 11. The output of the unitary U implementing the time-bound QPULBA model can be conditioned based on the required application. The last three cases presented in Section 3.3 conditions the programs with respect to the outputs, conditions the outputs with respect to a specific output, or conditions the final state with respect to a specific state. Thereafter, the conditioned register can be amplified using Grover search, or near-term variational quantum optimization approaches using parametric circuits such as QAOA [46–48]. Since the programs, outputs, and state qubits are entangled in a quantum associative memory, the amplitude of corresponding registers also evolves, such that the probability of measuring out the solutions increases. The results can then be sampled with high probability.

Estimation of algorithmic properties of a dataset can be very useful as it points to mechanistic connections between elements of a system, even those that do not yield any regular statistical patterns that can be captured with more traditional tools based on probability theory and information theory. Theoretical applications of algorithmic metrics such as Kolmogorov complexity is widespread [10], finding various uses in artificial general intelligence, theoretical physics, psychology, data compression, finance, linguistics, neuropsychology, psychiatry, genetics, sociology, behavioral sciences, image processing, among others. However, estimating algorithmic information for practical datasets is often computationally intractable due to the large number of enumerations that needs to be executed. The exploration in EAIT is growing in popularity due to its ubiquity in tasks that can be modeled as inductive reasoning. It connects theoretical computer science to the real-world by a quasi-empirical approach to mathematics (popularized as metamathematics by Gregory Chaitin). It involves enumerating and running programs to understand its statistical mechanics and is similar to Stephen Wolfram's computational universe approach [49].

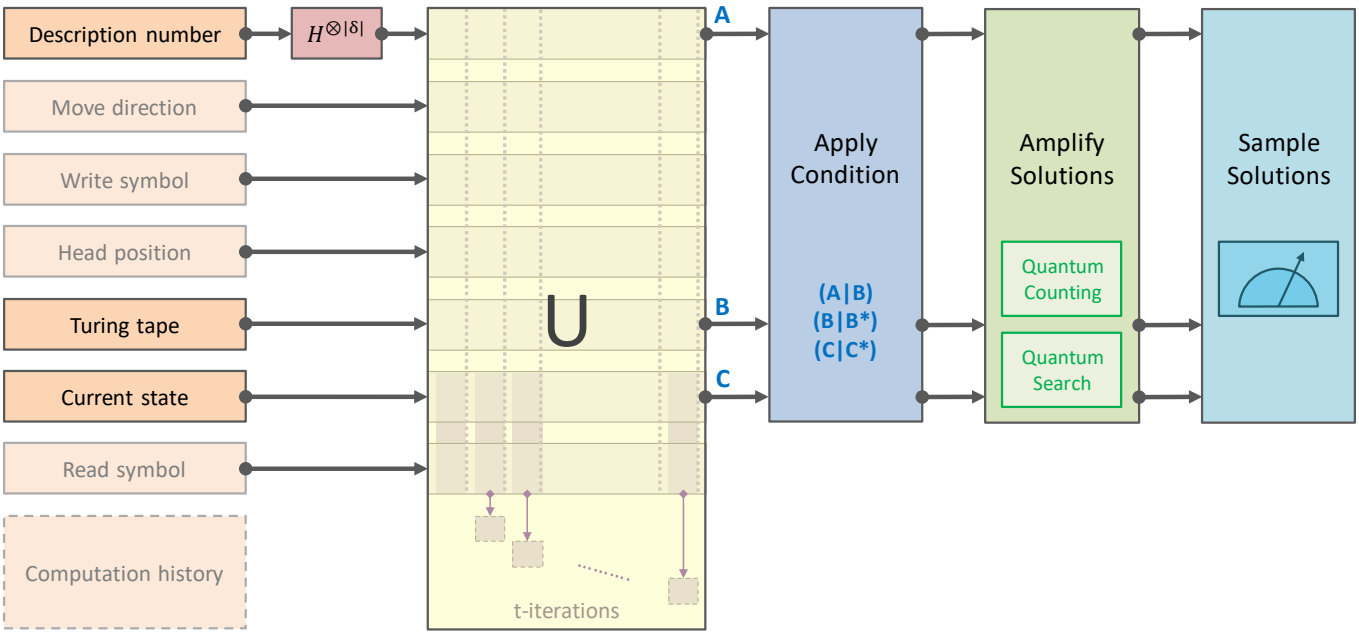


Figure 11. Application development framework for quantum experimental algorithmic information theory.

5.1. Meta-Biology and Artificial Life

Quantum EAIT can be used to accelerate meta-biology experiments in open-ended evolution [50] as busy beaver functions originally proposed in [1]. Meta-biology, as introduced by Gregory Chaitin, provides DNA linguistics [51–53] with an algorithmic information theory perspective, allowing exhaustive enumeration as an experimental method to understand the DNA as a code. In recent years, the field of algorithmic bioinformatics [54] was pioneered by multi-disciplinary research in Hector Zenil's group, achieving impressive results [27] using CTM and BDM. Our research will complement most of the work [55] in this field that uses classical algorithms, by considering a quantum accelerator extending to the same wide range of applications. Our implementations are currently executed on quantum computing simulators such as the QX or Qiskit due to the unavailability of quantum processors that meet the requirements of the multiplicity, connectivity and quality of qubits. While this limits our problem size due to the exponential overhead of simulation platforms, the high-level OpenQL and Qiskit programming language is generic and can target real quantum processors once they reach a better technology readiness level. This is facilitated by the abstraction layers of the quantum accelerator stack [56] as part of this research.

In our past research, we developed quantum algorithms for accelerating DNA sequence reconstruction using both alignment [57]- and assembly [58]-based approaches. This research targets the analysis phase, i.e., after the genome has been sequenced. Thus, from pattern matching, this work extends into the domain of pattern recognition and generation, for example in synthetic biology, xenobiology and minimal genome.

Here we review some recent developments in classical EAIT that can benefit significantly from our quantum approach when large-scale quantum computing reaches technological maturity.

5.2. Phylogenetic Tree Analysis Using EAIT

Recently, AIT techniques were successfully applied [59] in constructing the phylogenetic tree of RNA viruses. This research was conducted in the context of understanding the SARS-CoV-2 coronavirus responsible for the ongoing pandemic. Studying the genetic information by means of these computational tools can shed light on the rapid spread of SARS-CoV-2 and whether its evolution is driven by mutations, recombination or genetic variation. This information can then be applied for the development of diagnostic tools, effective antiviral therapies and in the understanding of viral diseases in general.

The diversity of viruses prevents reconstruction of evolutionary histories as they lack any equivalent set of universally conserved genes on which to construct a phylogeny, such as eucaryotes. Viruses differ significantly in their genetic material, RNA or DNA, and configurations (double or single stranded), as well as the orientation of their encoded gene. The research analyzed this genetic information by means of the Kolmogorov complexity and Shannon information theories. In the first case, the normalized information distance and the normalized compression distance is estimated using the zlib compressor. In the second, a statistical approach is adopted by constructing histograms for the relative frequency of base triplets and interpreted using entropy, cumulative residual entropy and Jensen–Shannon divergence.

The results indicate clearly the superior performance of the approaches based on the Kolmogorov complexity. The clusters are easily distinguishable and a relation is observed between the new SARS-CoV-2 virus and some CoV found in bats and pangolin, which are the most likely intermediate host from which the pandemic spread to humans.

This type of methodology may help to study how an animal virus jumped the boundaries of species to infect humans, and pinpoint its origin knowledge can help to prevent future zoonotic events. The statistical and computational techniques allow different perspectives over viral diseases that may be used to grasp the dynamics of the diseases. These methodologies may help interpreting future viral outbreaks and to provide addi-

tional information concerning these infectious agents and understand the dynamics of viral diseases.

5.3. Protein-Protein Interaction Analysis Using EAIT

There have been many successes in EAIT in recent years, especially in the field of biological sequence analysis. These techniques expand our understanding of the mechanisms underlying natural and artificial systems to offer new insights. Algorithmic information dynamics (AID) is at the intersection of computability, algorithmic information, dynamic systems, and algebraic graph theory to tackle some of the challenges of causation from a model-driven mechanistic perspective, in particular, in application to behavioral, evolutionary, and molecular reprogramming.

A recent exploration of this is in understanding the protein-protein interaction (PPI) map [60] between the SARS-CoV-2 proteins in human cells (the coronavirus responsible for the Covid-19 pandemic) and human proteins. 332 high-confidence PPI were experimentally identified using affinity-purification mass spectrometry. However, the mechanistic cause behind these specific interactions is not a well understood phenomena yet. A recent work [61] tries to explore the Kolmogorov complexity estimates of these PPI and found a positive correlation in the BDM values of the interactions. Such studies will help us predict in silico the biological dynamics, helping us find drug targets for therapeutics.

The BDM used for the study is based on the ACSS database, limited to the block length of 13. Extending ACSS to larger block lengths with help bridge the causal gap, which for longer strings such as proteins can be considerable, limiting its advantage over traditional entropy-based methods. The quantum framework described in this paper can potentially extend the ACSS database to empower the BDM more toward the actual CTM value.

5.4. In-Quanto Synthetic Biology

The ability to design new protein or DNA sequences with desired properties would revolutionize drug discovery, healthcare, and agriculture. However, this is challenging as the space of sequences is exponentially large and evaluating the fitness of proposed sequences requires costly wet-lab experiments. Ensemble approaches [62] with various machine learning methods, mostly generative adversarial networks (GAN), suitable to guide sequence design are employed in silico, instead of relying on wet-lab processes during algorithmic development. Since the mechanism for determining the fitness of the model is known, it can be encoded as a quantum kernel that evaluated in superposition the population of sequences. This is part of our future exploration in using the framework developed in this paper for in silico (or in quanto) synthetic biology.

6. Conclusions

In this research, we presented a framework for empirically evaluating algorithmic metrics on a quantum accelerator. The estimation of the universal prior distribution and thereby the algorithmic complexity and algorithmic probability of finite sequences is theoretically the most optimal technique for inferring algorithmic structure in data for discovering causal generative models. These metrics are uncomputable but can be approximated in practice by restricting the time and memory resources available to the computational model. Nevertheless, due to the exponential scaling of the number of possible automata that need to be enumerated they are intractable except for the simplest of the cases on classical computation. Moreover, owing to the unstructured output and computational irreducibility, it is not possible to dequantized or approximate the computation using heuristics. In this work, we propose a quantum circuit framework to estimate the universal distribution by simulating a superposition of programs (or transition functions) for a resource-bounded automata. The quantum resource complexity scales linearly in qubits and gates with respect to the data or automata size, thus achieving a polynomial speedup over classical exhaustive enumeration. Thus, exploring the space of program-output relations is one of the most promising problems for demonstrating speedup using quantum search on the

quantum supremacy roadmap. Specific properties of the program or output data can be inferred from the universal distribution represented as a quantum superposition.

The algorithmic information theoretic approach of causal generative mechanism discovery is more theoretically sound than data-driven approaches such as deep learning, lossless compression and Shannon entropy-based correlation, allowing it to find causal insights missed by these approaches. As a use case, we presented a full experimental framework for using this approach on DNA sequences for meta-biology and genome analysis. This is the first time a quantum computation approach is implemented for approximating algorithmic information. We implemented our copy-left design on the Qiskit programming language and tested it using the quantum computing simulator. The availability of better quantum processors would allow this algorithm to be readily ported on a quantum accelerator with our quantum computing stack. With quantum-accelerated genome analysis, a better understanding of the algorithmic structures in DNA sequences would greatly advance domains such as personalized medication and artificial life.

Author Contributions: Conceptualization, A.S. and Z.A.-A.; methodology, A.S. and Z.A.-A.; software, A.S.; validation, A.S. and Z.A.-A.; formal analysis, A.S. and Z.A.-A.; investigation, A.S. and Z.A.-A.; writing—original draft preparation, A.S.; writing—review and editing, A.S., Z.A.-A. and K.B.; visualization, A.S. and Z.A.-A.; supervision, Z.A.-A. and K.B.; project administration, Z.A.-A. and K.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable

Informed Consent Statement: Not applicable

Data Availability Statement: The data presented in this study are openly available on GitHub at DOI:10.5281/zenodo.4607476.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. QPULBA Quantum Circuit

An example quantum circuit for the 2 state 2 symbol 1 dimension taped QPULBA is presented here as a reference. While execution, the QPULBA is initialized and then undergoes the following unitary transformations for the pre-determined number of steps. These correspond to one step of a classical UTM, with the distinction of the computation now evolving in a superposition of all possible classical automata.

1. Read: $\{q_{read}\} \leftarrow U_{read}(\{q_{head}, q_{tape}\})$
2. Transition evaluation: $\{q_{write}, q_{ch}, q_{move}\} \leftarrow U_{\delta}(\{q_{read}, q_{state}, q_{\delta}\})$
3. Write: $\{q_{tape}\} \leftarrow U_{write}(\{q_{head}, q_{write}\})$
4. Move: $\{q_{head}\} \leftarrow U_{move}(\{q_{head}, q_{move}\})$
5. Reset

The initialization and each of the QPULBA steps are shown in the following circuits.

Initialize: For measuring the algorithmic probability or the universal distribution, all possible programs (represented by the transition table) need to be evolved in a superposition. All other qubits are kept at the ground or default state of $|0\rangle$. The circuit is shown in Figure A1.

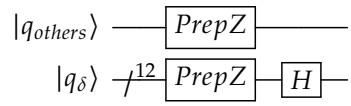


Figure A1. Initialization quantum circuit for QPULBA 2-2-1.

Read tape: The quantum circuit for read implements a multiplexer with the tape as the input signals and the binary coded head position as the selector lines, as shown in Figure A2. The read head is the output.

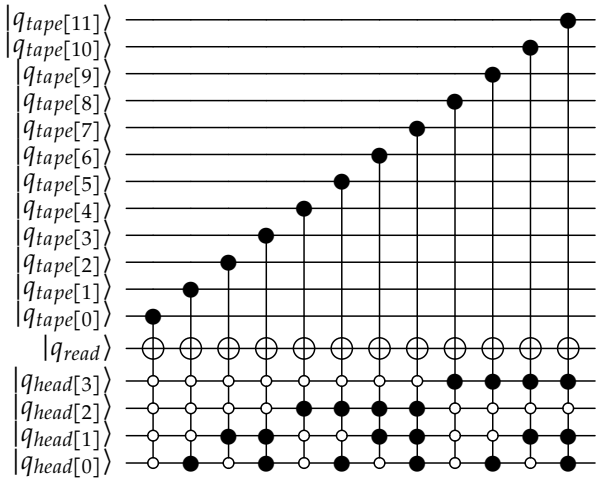


Figure A2. Read tape quantum circuit for QPULBA 2-2-1.

Transition table lookup: The transition table encoding is: $[Q_t|R_\Gamma] \rightarrow [Q_{t+1}|M_{l/r}|W_\Gamma]$. The transition function circuit is shown in Figure A3.

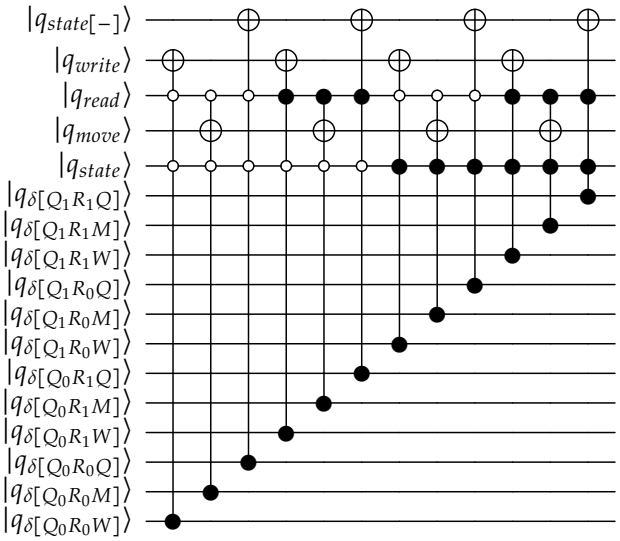
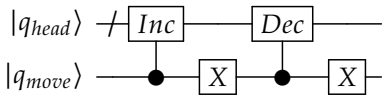


Figure A3. Transition function quantum circuit for QPULBA 2-2-1.

Write tape: The quantum circuit for write implements a de-multiplexer with the tape as the output signals and the head position as the selector lines, as shown in Figure A4. The write head is the input.

Move: There are many choices for implementing the move, e.g., a looped tape (overflow or underflow is ignored and trimmed), error flag is raised and halts, overflow/underflow is ignored, etc. Here, a looped tape is implemented. The head is incremented or decremented using the move qubit as control.



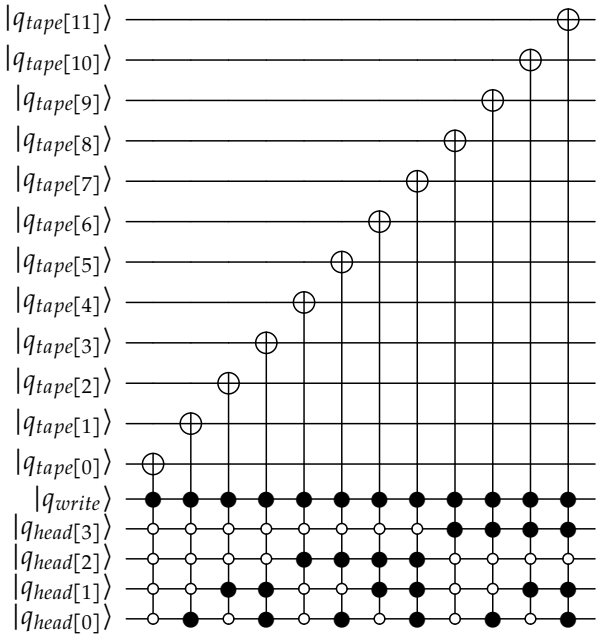


Figure A4. Write tape quantum circuit for QPULBA 2-2-1.

The increment/decrement circuit is a special case of the quantum full adder [63] with the first register, a set to 1. For the QPULBA 2-2-1 case, the length of the circular tape is 12, thus, the increment and decrement needs to be modulo 12. For increment, when the q_{head} equals 11, it should increment to $(11 + 1) \bmod 12 = 0$, while for decrement, (0–1) mod 12 = 11. Thus, for these edge cases, we need to increment/decrement by 5 instead of 1. We set the a_2 bit to change the effective value of a from $1 = 0001_2$ to $5 = 0101_2$ for the addition/subtraction. This operation is conditioned on the head value and move bit, and denoted as the overflow/underflow qubit $|ovfw\rangle/|udfw\rangle$. The a_2 bit is uncomputed based on the incremented value being 0 or the decremented value being 11.

The carry (C), sum (S) and reverse carry ($C^\dagger$) blocks are defined as follows:

```
.sum                .carry                  .reverse_carry
cnot A0,S0          toffoli A0,B0,C1        toffoli C0,B0,C1
cnot B0,S0          cnot A0,B0              cnot A0,B0
                    toffoli C0,B0,C1        toffoli A0,B0,C1
```

In this design, $c_3c_2c_1 = 000$ are 3 ancilla (c_0 is not required), $a_0 = q_{move}$, $a_3a_2a_1 = 000$, $b_3b_2b_1b_0 = q^3_{head}q^2_{head}q^1_{head}q^0_{head}$ and b_4 is ignored. The circuit in Figure A5 shows the 4-bit modulo-12 quantum increment circuit using the quantum adder blocks, while the decrement circuit is shown in Figure A6.

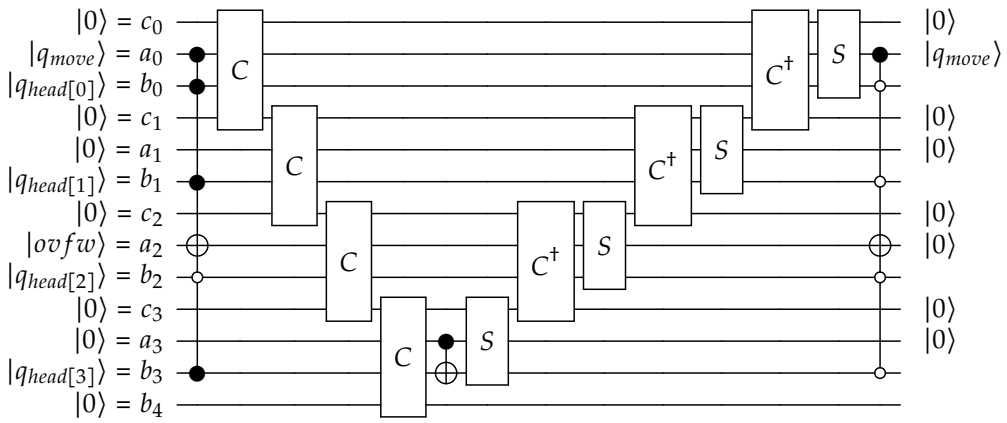


Figure A5. Modulo-12 quantum adder for implementing move tape head for QPULBA 2-2-1.

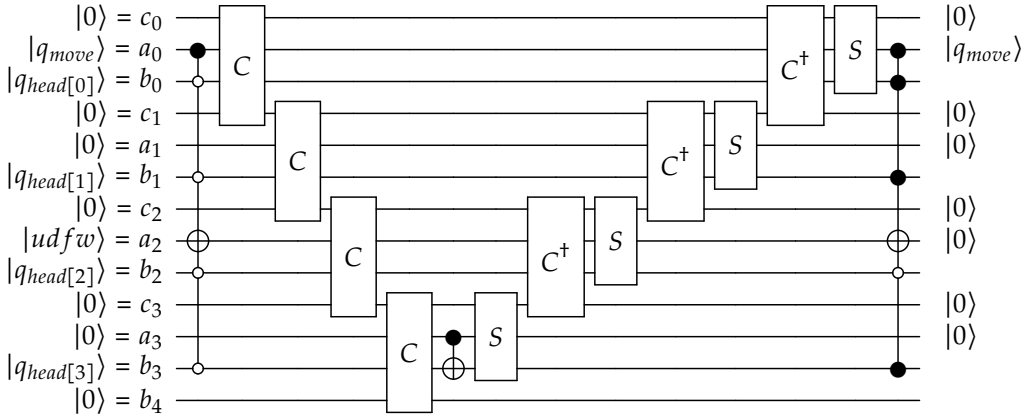


Figure A6. Modulo-12 quantum subtractor for implementing move tape head for QPULBA 2-2-1.

Both the increment and decrement circuits can be simplified considering some of the control qubits are always in the 0 state, so those CNOT/Toffoli gates can be ignored.

Reset: Quantum logic is universal and can implement any classical Boolean logic function using only the Toffoli (CCNOT) or Fredkin (CSWAP) gate. However, it is not always possible to uncompute all ancilla qubits. Specifically, if the function to be implemented is irreversible.

However, for the QPULBA case, we intend to evolve a superposition of all possible classical functions/programs C_f that can be represented by the description number encoding. These functions include both reversible as well as irreversible functions, thus, we cannot uncompute away the computation history of the state transition. Both the state and the read together preserve the evolution history. Thus, we need ancilla qubits in each step of the computation that would hold the transition history for the QPULBA. This limits the number of steps of the QPULBA we can implement or simulate. Besides the state and read, the write and move qubits need to be reset in each cycle. This is implemented by calling the FSM transition function once again with the previous state and the read.

References

1. Chaitin, G. *Proving Darwin: Making Biology Mathematical*; Vintage Books: Nwe York, NY, USA, 2012.
2. Brenner, S. Life's code script. *Nature* **2012**, *482*, 461–461.
3. Sarkar, A.; Al-Ars, Z.; Bertels, K. Quantum Accelerated Estimation of Algorithmic Information. *arXiv* **2020**, arXiv:2006.00987.
4. Neary, T.; Woods, D. Small weakly universal Turing machines. In *International Symposium on Fundamentals of Computation Theory*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 262–273.
5. Zenil, H.; Badillo, L.; Hernández-Orozco, S.; Hernández-Quiroz, F. Coding-theorem like behaviour and emergence of the universal distribution from resource-bounded algorithmic probability. *Int. J. Parallel, Emergent Distrib. Syst.* **2019**, *34*, 161–180.
6. Hutter, M. Algorithmic information theory: a brief non-technical guide to the field. *arXiv* **2007**, arXiv:cs/0703024.
7. Solomonoff, R.J. A formal theory of inductive inference. Part I. *Inf. Control* **1964**, *7*, 1–22.
8. Kolmogorov, A.N. Three approaches to the quantitative definition of information. *Int. J. Comput. Math.* **1968**, *2*, 157–168.
9. Chaitin, G.J. On the length of programs for computing finite binary sequences. *JACM* **1966**, *13*, 547–569.
10. Li, M.; Vitányi, P. *An Introduction to Kolmogorov Complexity and Its Applications*; Springer: Cham, Switzerland, 2008; Volume 3.
11. Levin, L.A. Laws of information conservation (nongrowth) and aspects of the foundation of probability theory. *Probl. Peredachi Informatsii* **1974**, *10*, 30–35.
12. Schmidhuber, J. The Speed Prior: A new simplicity measure yielding near-optimal computable predictions. *International Conference on Computational Learning Theory*; Springer: Berlin/Heidelberg, Germany, 2002; pp. 216–228.
13. Soler-Toscano, F.; Zenil, H.; Delahaye, J.P.; Gauvrit, N. Calculating Kolmogorov complexity from the output frequency distributions of small Turing machines. *PLoS ONE* **2014**, *9*, e96223.
14. Catt, E.; Hutter, M. A Gentle Introduction to Quantum Computing Algorithms with Applications to Universal Prediction. *arXiv* **2020**, arXiv:2005.03137.
15. Molina, A.; Watrous, J. Revisiting the simulation of quantum Turing machines by quantum circuits. *Proc. R. Soc. A* **2019**, *475*, 20180767.
16. Gorard, J. *Some Quantum Mechanical Properties of the Wolfram Model*; Complex Systems: Champaign, IL, USA, 2020; pp. 537–598..
17. Hollenberg, L.C. Fast quantum search algorithms in protein sequence comparisons: Quantum bioinformatics. *Phys. Rev. E* **2000**, *62*, 7532.

18. Hutter, M. *Universal Artificial Intelligence: Sequential Decisions Based on Algorithmic Probability*; Springer-Verlag: Berlin/Heidelberg, Germany, 2004.
19. Stanley, K.O.; Clune, J.; Lehman, J.; Miikkulainen, R. Designing neural networks through neuroevolution. *Nat. Mach. Intell.* **2019**, *1*, 24–35.
20. Chaitin, G.J. Meta math! the quest for omega. *arXiv* **2004**, arXiv:math/0404335.
21. Marletto, C. Constructor theory of life. *J. R. Soc. Interface* **2015**, *12*, 20141226.
22. Endoh, Y. Mame/Doublehelix. 2020. Available online: https://github.com/mame/doublehelix (accessed on 3 February 2021).
23. Sarkar, A.; Al-Ars, Z.; Bertels, K. Quines are the Fittest Programs-Nesting Algorithmic Probability Converges to Constructors. *Preprints* **2020**. doi:10.20944/preprints202010.0584.v1.
24. Noireaux, V.; Maeda, Y.T.; Libchaber, A. Development of an artificial cell, from self-organization to computation and self-reproduction. *Proc. Natl. Acad. Sci. USA* **2011**, *108*, 3473–3480.
25. Sipper, M.; Reggia, J.A. Go forth and replicate. *Sci. Am.* **2001**, *285*, 34–43.
26. Condon, A.; Kirchner, H.; Larivière, D.; Marshall, W.; Noireaux, V.; Tlusty, T.; Fourmentin, E. Will biologists become computer scientists? *EMBO Rep.* **2018**, *19*, e46628.
27. Zenil, H.; Minary, P. Training-free measures based on algorithmic probability identify high nucleosome occupancy in DNA sequences. *Nucleic Acids Res.* **2019**, *47*, e129.
28. Gauvrit, N.; Zenil, H.; Delahaye, J.P.; Soler-Toscano, F. Algorithmic complexity for short binary strings applied to psychology: A primer. *Behav. Res. Methods* **2014**, *46*, 732–744.
29. Zenil, H. Compression-based investigation of the dynamical properties of cellular automata and other systems. *arXiv* **2009**, arXiv:0910.4042.
30. Zenil, H.; Villarreal-Zapata, E. Asymptotic behavior and ratios of complexity in cellular automata. *Int. J. Bifurc. Chaos* **2013**, *23*, 1350159.
31. Brandouy, O.; Delahaye, J.P.; Ma, L.; Zenil, H. Algorithmic complexity of financial motions. *Res. Int. Bus. Financ.* **2014**, *30*, 336–347.
32. Zenil, H.; Delahaye, J.P. An algorithmic information theoretic approach to the behaviour of financial markets. *J. Econ. Surv.* **2011**, *25*, 431–463.
33. Delahaye, J.P.; Zenil, H. On the Kolmogorov-Chaitin Complexity for short sequences. *arXiv* **2007**, arXiv:abs/0704.1043.
34. Gauvrit, N.; Singmann, H.; Soler Toscano, F.; Zenil, H. Algorithmic Complexity for Short Strings [R Package Acss Version 0.2-5]. Available online: https://cran.r-project.org/web/packages/acss/index.html (accessed on 3 February 2021).
35. Calude, C.S.; Stay, M.A. Most Programs Stop Quickly or Never Halt. *arXiv* **2006**, arXiv:cs/0610153.
36. Valiant, L.G. The complexity of computing the permanent. *Theor. Comput. Sci.* **1979**, *8*, 189–201.
37. Complexity Zoo. Available online: https://complexityzoo.net/Complexity_Zoo (accessed on 3 February 2021).
38. Aaronson, S. Quantum computing, postselection, and probabilistic polynomial-time. *Proc. R. Soc. A Math. Phys. Eng. Sci.* **2005**, *461*, 3473–3482.
39. Aaronson, S.; Arkhipov, A. The computational complexity of linear optics. In Proceedings of the 43rd Annual ACM Symposium on Theory of Computing, San Jose, CA, USA, 6–8 June 2011; pp. 333–342.
40. Abrams, D.S.; Lloyd, S. Simulation of many-body Fermi systems on a universal quantum computer. *Phys. Rev. Lett.* **1997**, *79*, 2586.
41. Zhong, H.S.; Wang, H.; Deng, Y.H.; Chen, M.C.; Peng, L.C.; Luo, Y.H.; Qin, J.; Wu, D.; Ding, X.; Hu, Y.; et al. Quantum computational advantage using photons. *Science* **2020**, *370*, 1460–1463.
42. Brassard, G.; Høyer, P.; Tapp, A. Quantum counting. In *International Colloquium on Automata, Languages, and Programming*; Springer: Berlin/Heidelberg, Germany, 1998; pp. 820–831.
43. Wie, C.R. Simpler quantum counting. *arXiv* **2019**, arXiv:1907.08119.
44. Suzuki, Y.; Uno, S.; Raymond, R.; Tanaka, T.; Onodera, T.; Yamamoto, N. Amplitude estimation without phase estimation. *Quantum Inf. Process.* **2020**, *19*, 75.
45. Aaronson, S.; Rall, P. Quantum approximate counting, simplified. In Proceedings of the Symposium on Simplicity in Algorithms, Salt Lake City, UT, USA, 6–7 January 2020; pp. 24–32.
46. Farhi, E.; Goldstone, J.; Gutmann, S. A quantum approximate optimization algorithm. *arXiv* **2014**, arXiv:1411.4028.
47. Jiang, Z.; Rieffel, e.g.,; Wang, Z. Near-optimal quantum circuit for Grover's unstructured search using a transverse field. *Phys. Rev. A* **2017**, *95*, 062317.
48. Morales, M.E.; Tlyachev, T.; Biamonte, J. Variational learning of Grover's quantum search algorithm. *Phys. Rev. A* **2018**, *98*, 062333.
49. Wolfram, S. *A New Kind of Science*; Wolfram Media: Champaign, IL, USA, 2002; Volume 5.
50. Adams, A.; Zenil, H.; Davies, P.C.; Walker, S.I. Formal definitions of unbounded evolution and innovation reveal universal mechanisms for open-ended evolution in dynamical systems. *Sci. Rep.* **2017**, *7*, 1–15.
51. Atlan, H.; Koppel, M. The cellular computer DNA: program or data. *Bull. Math. Biol.* **1990**, *52*, 335–348.
52. Dong, S.; Searls, D.B. Gene structure prediction by linguistic methods. *Genomics* **1994**, *23*, 540–551.
53. Coste, F. Learning the language of biological sequences. In *Topics in Grammatical Inference*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 215–247.

54. Vallejo, E.E.; Ramos, F. Evolving Turing machines for biosequence recognition and analysis. In *European Conference on Genetic Programming*; Springer: Berlin/Heidelberg, Germany, 2001; pp. 192–203.
55. Antão, R.; Mota, A.; Machado, J.T. Kolmogorov complexity as a data similarity metric: Application in mitochondrial DNA. *Nonlinear Dyn.* **2018**, *93*, 1059–1071.
56. Bertels, K.; Ashraf, I.; Nane, R.; Fu, X.; Riesebos, L.; Varsamopoulos, S.; Mouedenne, A.; Van Someren, H.; Sarkar, A.; Khammassi, N. Quantum computer architecture: Towards full-stack quantum accelerators. *arXiv* **2019**, arXiv:1903.09575.
57. Sarkar, A.; Al-Ars, Z.; Almudever, C.G.; Bertels, K. An algorithm for DNA read alignment on quantum accelerators. *arXiv* **2019**, arXiv:1909.05563.
58. Sarkar, A.; Al-Ars, Z.; Bertels, K. QuASeR–Quantum Accelerated De Novo DNA Sequence Reconstruction. *arXiv* **2020**, arXiv:2004.05078.
59. Machado, J.T.; Rocha-Neves, J.M.; Andrade, J.P. Computational analysis of the SARS-CoV-2 and other viruses based on the Kolmogorov's complexity and Shannon's information theories. *Nonlinear Dyn.* **2020**, *101*, 1731–1750.
60. Gordon, D.E.; Jang, G.M.; Bouhaddou, M.; Xu, J.; Obernier, K.; White, K.M.; O'Meara, M.J.; Rezelj, V.V.; Guo, J.Z.; Swaney, D.L.; et al. A SARS-CoV-2 protein interaction map reveals targets for drug repurposing. *Nature* **2020**, *583*, 459–468.
61. Adams, A. The Role of Emergence in Open-ended Systems. *AUTOMATA* **2020**, unpublished.
62. Angermueller, C.; Belanger, D.; Gane, A.; Mariet, Z.; Dohan, D.; Murphy, K.; Colwell, L.; Sculley, D. Population-Based Black-Box Optimization for Biological Sequence Design. *arXiv* **2020**, arXiv:2006.03227.
63. Vedral, V.; Barenco, A.; Ekert, A. Quantum networks for elementary arithmetic operations. *Phys. Rev. A* **1996**, *54*, 147.