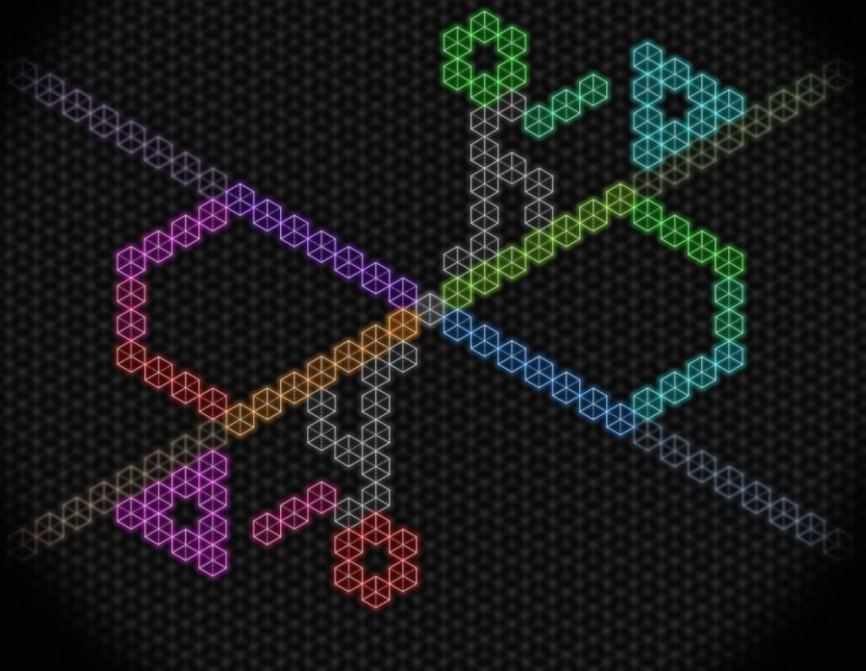
Applications of Quantum Computation and Algorithmic Information

for Causal Modeling in Genomics and Reinforcement Learning



ARITRA SARKAR

APPLICATIONS OF QUANTUM COMPUTATION AND ALGORITHMIC INFORMATION

FOR CAUSAL MODELING
IN GENOMICS AND REINFORCEMENT LEARNING

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus prof.dr.ir. T.H.J.J. van der Hagen
chair of the Board for Doctorates
to be defended publicly on
Monday 11 July 2022 at 17:30 hours

by

Aritra SARKAR

Master of Science in Computer Engineering,
Delft University of Technology, The Netherlands,
born in Purulia, India

This dissertation has been approved by the promotors.

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof. dr. K.L.M. Bertels,	Delft University of Technology, promotor
Dr. ir. Z. Al-Ars,	Delft University of Technology, promotor
Independent members:	
Prof. dr. ir. M. Hutter	Australian National University, Australia
Prof. dr. H.M. Buhrman	University of Amsterdam, The Netherlands
Prof. dr. D. DiVincenzo	RWTH Aachen University, Germany
	Delft University of Technology
Prof. dr. ir. L. DiCarlo	Delft University of Technology
Prof. dr. ir. H.P. Hofstee	Delft University of Technology

Keywords: quantum algorithms, genomics, algorithmic information theory, reinforcement learning

Printed by: Ridderprint

Front & Back: Front cover designed with help from Rupsa Chakraborti. It is an artistic depiction of algorithmic absurdism (refer to epilogue). It illustrates two sentient agents embedded on a space-time diagram, rolling an impossible tribar into the future/past, akin to the absurd metaphor of Sisyphus. Ad hoc algorithmic meaning and paradoxes emerge from their autopoietic existence, perception, actuation, and computational capabilities.

ISBN 978-94-6366-530-8

An electronic version of this dissertation is available at https://repository.tudelft.nl/.

to less-sentient present arrangements of information . . .
that cannot yet/any-longer comprehend the ideas presented herein

Contents

Acknowledgements

Foremost, I would like to express my sincere gratitude to my promoters, *Koen Bertels* and *Zaid Al-Ars*. I remember timidly stepping into Koen's office in 2017 to ask if he was willing to promote my master's thesis. I had prepared a list for this conversation on how I could fit into the ongoing research of the Quantum Computer Architecture (QCA) team. After chatting about my background and the ideas in that list, he proposed to me, 'keeping aside the research going on in the group, if given complete freedom to propose a new topic, what would you want to do?' I impulsively exclaimed that I wanted to work on designing new quantum algorithms. From my computer applications perspective, the design and benchmarking of the quantum stack used unrepresentative and unpragmatic algorithms. Koen agreed! While that was the connection to QCA, our goal had a broader vision. Soon we would have large quantum computers; however, their real-world use cases, besides the canonical examples, were still vague. Discovering high-impact use cases of quantum computation was the goal that Koen laid out for me.

While brainstorming applications for starting my exploration, I tried my luck mentioning BlueBee, a high-performance genomics company that Koen and Zaid founded. Unbeknownst to me, in that context, they were drafting a project proposal – to explore the possibility of quantum acceleration for genomics. Analyzing biology as a structured code fascinated me. I contentedly agreed to explore this terra incognita . . . starting my research career in quantum computing. Besides their technical guidance, over these years, Koen and Zaid have imbued in me many soft skills that made me a better researcher. Thank you, Koen, for insisting on understandable technical presentations and inspiring me, like countless others, of quantum computing's impact. Thank you, Zaid, for hand-holding me into the art of scientific writing, teaching me to evaluate and decide in critical situations practically, and for all those digressing chats on the fringe frontiers of science and technology.

I would like to convey my thanks to *Carmen Almudever* (Carmina) for being the daily supervisor for my master's research. Your support, both technically and morally, was indispensable for the work. *Imran Ashraf*, without your expert aid in OpenQL, QX, and all the other auxiliary frameworks, those quantum algorithms could never have been implemented. Special thanks to my doctoral committee for reading this thesis and giving me valuable comments to improve it. Over these years, I have shared (physical and virtual) offices and memories with many. *Andreas Papageorgiou, Amitabh Yadav, Leon Riesebos, Radouane Oudrhiri, Hans van Someren, Matthias Möller, Abid Moueddene, Xiang Fu, Lingling Lao, Savvas Varsamopoulos, Yasaman Samadi, Tamara Sarac, Yassine Ferjani, Kourosh Dogahe, Hongfeng Zhang, João Vieira, Smaran Adarsh, Koen Mesman, Hubb Donkers, Daniel Manzano, Nader Khammassi, Jeroen van Straten, Sebastian Feld, Medina Bandić, Erik de Vries, Matthijs Brobbel, Razvan Nane, Diogo Valada, Mengyu Zhang, Miguel Moreira, Vieri Mattei, Alejandro Morais, Yaoling Yang, Nikiforos Paraskevopoulos, Smit Chaudhary, Peter Elgar, Matti Dreef, Folkert de Ronde, Luise Prielinger, Felix Paul,*

...thanks to all those nice conversations and fun during the group dinners, pub crawls, carting, laser tags, etc. I would particularly like to thank *Thomas Hubregsen* (Tom) and *Matthew Steinberg* (Matt) for all those enriching brainstorming sessions on quantum information. I also want to thank the students for whom I was the technical guide. *Anna Maria Krol* (Anneriet), *Neil Eelman*, *Duarte Frazão*, *Ravish Budhrani*, *Karthik Krishnakumar*, *Harshitta Gandhi*, *Delphine De Smedt*, I have learned a lot from you all. Anneriet, thank you for translating the summary of this dissertation into Dutch. *Joyce*, *Lidwina*, *Marja*, *Trisha*, *Paul*, thanks for all your administrative help. Thanks to my other colleagues at the Department of Quantum & Computer Engineering and QuTech. I hope the future holds many more collaborations with each one of these brilliant minds. You gave me a conducive atmosphere to conduct my doctoral research and be part of a visionary team.

I started my journey into artificial intelligence and quantum computing via the online courses from *Sebastian Thrun*, *Andrew Ng*, *Peter Norvig*, *Umesh Vazirani* and *Leonard Susskind*. At TU Delft, this knowledge was reinforced by the excellent courses in quantum from *Leonardo DiCarlo*, *Stephanie Wehner*, *Thomas Vidick*, *Carmen Almudever*, *Fabio Sebastiano* and *David Elkouss*. During my PhD, I took some fascinating courses in quantum machine learning, algorithmic information, computational genomics and artificial intelligence from *Peter Wittek*, *Hector Zenil*, *Pavel Pevzner*, *Phillip Compeau* and *Jean-Louis Dessalles*. All these courses gifted me the oars to navigate the sea of knowledge.

It is unfathomable to list all possible scientists who laid the path for this research. My favorites among them, however, are *John Archibald Wheeler*, *John von Neumann* and *Ray Solomonoff*. Tracing back, the ideas in this dissertation would inevitably converge on one of their work. *Jürgen Schmidhuber*'s Gödel machines, *Gregory Chaitin*'s metabiology, *Hector Zenil*'s experimental AIT, *Marcus Hutter*'s AIXI, *David Deutsch*'s constructor theory, *Marcus Müller*'s QM based on Solomonoff induction, *Leonard Susskind*'s uncomplexity, *Stephen Wolfram*'s metamathematics, are some ideas that influenced my research considerably. I did not find much time for casual reading in the last few years. While dust settled on many books, I did (re)read some thought-provoking works of *Charles Dodgson*, *Jorge Luis Borges*, *Noson Yanofsky*, *Douglas Hofstadter*, *Michael Crichton*, *Vlatko Vedran*, *Amanda Gefter*, *Carlo Rovelli*, *Judea Pearl*, *Shashi Tharoor*, *Jiddu Krishnamurti*, and *Madhvācārya*. The philosophical aspects enriched my technical ideas.

I would not be here without the tutoring from some excellent teachers I was lucky to have had, my home tutors, at school, and at the two universities I attended. Thank you for building the foundation on which I now build my career. My colleagues and mentors at ISRO and TCSRI, thank you for teaching me the basics of research, project management, and the frontiers of space technology and cognitive robotics. I hope robotics and space engineering will intertwine and feature again in my future research. Special thanks to *B.S. Manoj* sir for guiding my bachelor thesis and eventually helping me in deciding to do a Ph.D. *R. Srividhya* ma'am, thank you for being the go-to person for evaluating my career decisions, for encouraging me to step outside my comfort zone, and for your support and belief in me ever since we met.

Over the last few years, I have also grown fond of creating awareness of quantum computing in the broader community. Thanks to Tom (PushQuantum), *Eràn Rotman*

(QC4HC), *Araceli Venegas-Gomez* (QURECA), and *Eamonn Darcy* (QUECWA) for providing me with the platform to do so. QWorld and its QIndia cousin are two communities that deserve special thanks. The QIntern 2021 event was a huge success, and the QKSA project found some adept collaborators. Thanks you *Adam Glos, Zoltan Zimboras, Bao Bach, Sarang Gosavi, Tamal Acharya* and all the interns who I worked with for the project, and thereafter in QResearch. QIndia has been a globally connected network with a shared zeal to popularize quantum research in India. It is always hard to believe that I have never physically met anyone yet. Thank you *Rishi Sreedhar, Akash Kundu, Achintya Paradkar, Anuj Mehrotra, Rajiv Krishnakumar, Rijul Sachdeva, Turbasu Chatterjee, Ameya Nambisan*, and all the other core team and participating members.

I want to thank my friends here in Delft and otherwise, with whom I could share the sparks of crazy inklings and the frustrations at various points in this journey. Specifically, thank you *Abhishek, Charu, Srinidhi, Mitali, Mitul*, for being there when I needed you the most.

I will forever be in debt to my roots in Purulia, Barunhat, and Kolkata. To all my relatives and in-laws, thanks for nurturing me and giving me the feeling of belonging. Thank you for all the scrumptious food and the warmth of your blessings. Especially, thank you *mama* (maternal uncle) for guiding me through major career decisions since childhood. Wherever you are *dadu* (grandpa), I hope I am progressing towards your wish that your grandsons would be at the international frontiers in their careers.

An honest acknowledgment should mention the online game, Defense of the Ancients (DotA), the activity I spent most of my time over the last decade when not involved in my studies or socializing. The OpenAI Five's achievement made me realize how far we have come in AI.

Like Schrödinger's cat in a box, the concoction of a doctoral researcher and a pandemic is a pernicious situation. Thanks to everyone who helped me keep my resolve and be resilient, both physically and mentally, to the effects of this real-life thought experiment.

It was also in the first year that I married my childhood love, *Rupsa*. Our 11 years of long-distance relationship eventually led to our first home here in Delft. Thanks for tolerating my tantrums ... and, most importantly, keeping me grounded on the fact that the topics of this dissertation are a very insignificant part of my existence and human society in general. From being captive at home for months during the pandemic to visiting cities around Europe, from snowball fights to mastering the Indian cuisine – I bet life has many more such adventures in store for us. I hope we will stay maximally entangled together forever!

Last but not the least, *baba & ma*, thank you for being the pillars of my life in this world. Thanks for taking care of me, educating me, sacrificing so much for me, listening to my everyday stories, protecting and advising me, pushing me to pursue my dreams ... a never-ending list that would still not do justice.

Completing a Ph.D. is not an easy task. However, these people (and many more I regrettably and unintentionally missed) made this journey an enjoyable, unforgettable, and fulfilling experience. Thank you!

Summary

Efforts to realize a sufficiently large controllable quantum processor are actively being pursued globally. These quantum devices are programmed by specifying the manipulation of quantum information via quantum algorithms. This doctoral research provides an application perspective to the design requirements of a quantum accelerator architecture. Early quantum algorithms focused specifically on the theoretical study of the benefits in computational resource cost by harnessing quantum phenomena. With small-scale quantum processors being available, the focus is now towards applying quantum algorithms to develop applications with high impact in societal, industrial, and scientific fields. No quantum devices exist that can execute quantum algorithms that can demonstrate a provable advantage for a real world use case. However, a proof of concept application pipeline can be demonstrated on a simulator framework. The research question of this dissertation is to identify high-impact long-term applications of quantum computation and formulate the corresponding logic. Three specific use cases are studied.

Use case 1 involves 'quantum-accelerated genome sequence reconstruction'. Faster sequencing pipeline would enable novel downstream applications like personalized medical treatment. Two different reconstruction methods, ab initio reference alignment, and de novo read assembly, are studied to identify the computational bottleneck. Corresponding quantum techniques are formulated, based on quantum search and heuristic quantum optimization, respectively. A new algorithm, quantum indexed bidirectional associative memory (QiBAM), is explicitly designed to address the requirements for approximate alignment of DNA sequences. We also proposed the quantum accelerated sequence reconstruction (QuASeR) strategy to perform de novo assembly. This is formulated as a QUBO and solved using QAOA on a gate-model simulator, as well as, on a quantum annealer.

Use case 2 involves 'quantum automata for algorithmic information'. A framework for causal inference based on algorithmic generative models is developed. This technique of quantum-accelerated experimental algorithmic information theory (QEAIT) can be ubiquitously applied to diverse domains. Specifically for genome analysis, the problem of identifying bit strings capable of self-replication is presented. We developed a new quantum circuit design of a quantum parallel universal linear bounded automata (QPULBA) model. This enables a superposition of classical models/programs to be executed, and their properties can be explored. The automaton prepares the universal distribution as a quantum superposition state which can be queried to estimate algorithmic properties of the causal model.

Use case 3 involves 'universal reinforcement learning in quantum environments'. This theoretical framework can be applied for automated scientific modeling. A universal artificial general intelligence formalism is presented that can model quantum processes. The developed quantum knowledge seeking agent (QKSA) is an evolutionary general reinforcement learning model for recursive self-improvement. It uses resource-bounded algorithmic complexity of quantum process tomography algorithms. The cost function determining the optimal strategy is implemented as a mutating gene within a quine. The utility function for an individual agent is based on a selected quantum distance measure between the predicted and perceived environment.

This dissertation researches foundational techniques and develops innovative applications of quantum computation and algorithmic information. These were applied specifically for causal modeling in genomics and reinforcement learning. Further exploration of the synergies among these interdisciplinary concepts would improve our understanding of various scientific disciplines like computation, intelligence, life, and cosmology.

SAMENVATTING

Over de hele wereld wordt er hard gewerkt aan het realiseren van een voldoende grote bestuurbare quantumprocessor. Deze quantum apparaten worden geprogrammeerd door het specificeren van de manipulatie van quantum informatie met quantum algoritmes. Dit promotieonderzoek biedt een perspectief op de ontwerpeisen voor een quantum acceleratie architectuur vanuit toepassingen voor zulke quantumprocessoren. Onderzoek naar quantum algoritmes legde in het begin voornamelijk de focus op het theoretische voordeel in rekenkracht, als quantumverschijnselen gebruikt werden. Nu er kleinschalige quantumprocessoren beschikbaar zijn, is de focus verschoven naar het toepassen van quantum algoritmes om applicaties te ontwikkelen met een grote impact op maatschappelijk, industrieel en wetenschappelijk vlak. Er bestaan nog geen quantum apparaten die een quantum algoritme kunnen uitvoeren dat een bewijsbaar voordeel heeft in de echte wereld. Maar een bewijs van concept voor een applicatie pijplijn kan gedemonstreerd worden met een simulatie framework. De onderzoeksvraag van dit proefschrift is het identificeren van toepassingen voor quantum computers die een hoge impact kunnen bieden op de lange termijn, en het formulieren van de bijbehorende logica. Drie specifieke toepassingen zijn uiteengezet.

De eerste toepassing betreft ‘quantum versnelde reconstructie van genoomsequenties’. Een snellere sequentie-pijplijn kan tot gevolg hebben dat andere toepassingen mogelijk worden, zoals gepersonaliseerde persoonlijke behandelingen. Twee verschillende reconstructie methodes zijn bestudeerd om de knelpunten te vinden: ab initio referentie uitlijning en de novo leesmontage. Voor elk wordt een quantum techniek geformuleerd, gebaseerd op respectievelijk quantum zoeken en heuristische quantum optimalisatie. Een nieuw algoritme, quantumgeïndexeerd bidirectioneel associatief geheugen (QiBAM), is ontworpen om aan de eisen voor benaderende uitlijning van DNA-sequenties te kunnen voldoen. Ook stellen we een strategie voor om de novo-assemblage uit te voeren: quantum versnelde sequentiereconstructie (QuASeR). De strategie is geformuleerd als een QUBO en wordt opgelost met behulp van QAOA op een gate-model simulator, evenals op een quantum annealer.

De tweede toepassing betreft ‘quantum automaten voor algoritmische informatie’. Er is een raamwerk ontwikkeld voor causale interferentie gebaseerd op algoritmische generatieve modellen. Deze techniek voor quantum versnelde experimentele algoritmische informatietheorie (QEAIT) kan alom worden toegepast in diverse domeinen. Specifiek voor genoomanalyse wordt het probleem gepresenteerd van het identificeren van bitstrings die in staat zijn zichzelf te repliceren. We hebben een nieuw quantum circuit ontwerp ontwikkeld voor modellen van quantum parallel universeel lineair begrensde automaten (QPULBA). Hierdoor wordt superpositie van klassieke modellen/programma’s mogelijk, en kunnen hun eigenschappen worden onderzocht. De automaat maakt van de universele distributie een quantum superpositietoestand, die kan worden gebruikt om een schatting te maken van de algoritmische eigenschappen van het causale model.

De derde toepassing betreft 'universeel versterkend leren in quantum omgevingen'. Dit theoretische framework kan worden toegepast voor geautomatiseerd wetenschappelijke modellering. Er wordt een formalisme gepresenteerd voor een universele kunstmatige algemene intelligentie, die quantum processen kan modelleren. De ontwikkelde quantum kenniszoekend agent (QKSA) is een evolutionair algemeen bekrachtigend leermodel voor recursieve zelfverbetering. De agent gebruikt de middelgebonden algoritmische complexiteit van quantum proces tomografie algoritmes. De kostfunctie die de optimale strategie bepaald is geïmplementeerd als een muterend gen in een quine. De utiliteitsfunctie voor een individuele agent is gebaseerd op een geselecteerde quantum afstandsmeeting tussen de verwachte en de waargenomen omgeving.

Dit proefschrift onderzoekt de fundementele technieken van quantum computatie en algoritmische informatie, en ontwikkeld daarvoor innovatieve toepassingen. Deze werden specifiek toegepast op causale modellering van genomen en bekrachtigend leren. Verdere verkenning van de synergie tussen deze interdisciplinaire concepten zou ons begrip verbeteren van verschiedene wetenschappelijke disciplines zoals compuatie, intelligente, leven en cosmologie.

1

APPLICATIONS OF QUANTUM COMPUTING

Quantum computing uses the laws of quantum mechanics to achieve an information processing advantage. Efforts to realize a sufficiently large controllable, and programmable quantum computer are actively being pursued globally. The specific goal of this research pertains to various applications and algorithm designs that target these computers. At the time of this research (2018-2022), no quantum devices exist that can execute quantum algorithms that demonstrate a provable advantage for a real world use case. However, quantum computing simulators running on classical computers can be used to validate the implementation and the impact on the application. The research question of this dissertation is to **identify high-impact long-term applications of quantum computation and formulate corresponding quantum algorithms to compute them**.

While the specific research goal of this dissertation pertains to the algorithm layer of the quantum computing stack, the doctoral research was carried out within a close-knit team. In such early days of this field, it is crucial to define a viable and promising research roadmap. Thus, an integral part of the doctoral research was to provide an application perspective to the design of a quantum accelerator. This chapter provides the background of quantum computation in general, and explains how the different research motivations and current signs of progress are used in defining the research methodology for the work presented in the later chapters.

This chapter is based on the following:

- Bertels, K., **Sarkar, A.**, & Ashraf, I. (2021). Quantum Computing—From NISQ to PISQ. *IEEE Micro, 41*(5), 24-32.
- Bertels, K., **Sarkar, A.**, Hubregtsen, T., Serrao, M., Mouedenne, A. A., Yadav, A., ... & Ashraf, I. (2020, March). Quantum computer architecture: Towards full-stack quantum accelerators. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (pp. 1-6). IEEE.
- Hubregtsen, T., Segler, C., Pichlmeier, J., **Sarkar, A.**, Gabor, T., & Bertels, K. (2020, March). Integration and Evaluation of Quantum Accelerators for Data-Driven User Functions. In *2020 21st International Symposium on Quality Electronic Design (ISQED)* (pp. 329-334). IEEE.

1.1. COMPUTATION: A HISTORIC PERSPECTIVE

Marveling on the pervasive and magnificent civilization that human beings have established, much of its incredible engineering genius has been made possible by precise computations. But even before the advent of the space age, the computer age, or even the industrial age, humanity had firmly seated itself at the epitome of evolutionary success. Use of tools to aid in various activities is arguably an essential aspect of intelligent behavior, which very few animals exhibit. While tools, in general, can mean anything from fire to wheels, we shall, in this section, have a brief replay of the evolution of computing methods over the ages - ultimately converging on the current challenges that motivate us for further research in this domain.

The widely used decimal system of counting is mostly attributed to our ten fingers. While this method of computation is limited to junior schools and might not seem a cognitive spectacle, monkeys sharing ~ 99% of our genetic codes cannot do any of those simple multiplication tables. Counting on fingers requires some amount of memorization. An easier method for early humans have been marks on cave walls or counting with stones. The abacus and such a pebble heap are not much different in concept. As civilization progressed, analog and mechanical computing devices like the Antikythera mechanism or the astrolabe were invented as special-purpose tools to aid calculation. A more general-purpose calculator was invented in the 17th century as the slide rule. Eventually, more autonomy was added as the machinery became more complex, like Pascal's calculator or Jacquard's loom. The loom wove pattern controlled by a paper tape constructed from punched cards. The cards could be changed without altering the mechanical design of the loom - a landmark achievement in programmability. Charles Babbage conceptualized and invented the first mechanical computer, the difference engine, in the early 19th century, to aid in navigational calculations. Thereafter, he realized a much more general design called the analytical engine. The input programs and data were provided via punched cards, while output devices included a printer, a curve plotter, and a bell. It incorporated a decimal arithmetic logic unit (ALU), control flow in the form of conditional branching and loops, and integrated memory; making it the first design for a general-purpose computer that could be described in modern terms as Turing-complete. Though the machine was not realized, Ada Lovelace, often regarded as the first computer programmer, recognized its applications and published the first algorithms for it. Currently, we are in a similar early stage for quantum computation.

The era of modern computing began with a flurry of developments concerning World War II. These devices were electro-mechanical, having low operating speeds. They were eventually superseded by much faster all-electric computers, initially using vacuum tubes. In 1941, Konrad Zuse built Z3, the world's first working electro-mechanical programmable, fully automatic digital computer. Replacement of the hard-to-implement decimal system by the simpler binary system added to the reliability. Purely electronic circuit elements gradually replaced their electromechanical equivalents in the early computers like Colossus, ENIAC, and EDVAC. As further innovation progressed, some of the architectural concepts seen in today's computers became established. These included the pervasive Boolean algebra for the ALU, instruction-sets for the microcode, and the concept of firmware (for device drivers) and the operating system. Another significant milestone is the concept of stored program architecture introduced in 1945 by

John von Neumann. This architecture evolved to be associated with a common bus to the random-access memory (RAM), preventing instruction fetch and a data operation simultaneously. The alternate more complex Harvard architecture has a dedicated address bus and memory, for data and instructions. The next generation of devices featured silicon-based junction (later, field-effect) transistors in the 1950s. These were smaller and more reliable while requiring less power than vacuum tubes, therefore dissipated less heat. What followed was an unprecedented boom in the electronics industry, bringing with it new applications and dependence on computation-heavy insights.

It is observed [1] that the single-thread performance, operating frequency, and power stagnated around 2005, shifting the trend to multi-core processors. The power consumption of a transistor is attributed to either the leakage current, the switching current or the static power. Thus, an exponential increase in frequency (the main driver for performance increase before 2005) also gave an exponential increase in power, limiting the number of transistors that can be kept powered on per unit chip area. At the architectural level, pipelining by instruction-level parallelism and speculative execution reached saturation, resulting in the adoption of explicit parallelism. However, the speedup a program can harness in a multi-core environment is limited by the parallel fraction of a program, given by Amdahl's law. Not all programs have a structure that can be beneficially parallelized, taking into account the communication and synchronizing overheads among the parts. Thus, the transition to specialized architectures for specific algorithmic structures becomes necessary. Finally, the advances in memory and processor-memory interconnect technology did not reach the same level of performance as the processors, increasing the access stalls between each level of cached memory hardware.

The shift to multi-core alleviated the computing industry, but only for a few years. The drawbacks of effectively utilizing multiple cores kept diminishing the performance returns for the engineering investment. The next generation of processors included various trade-offs, like dim-silicon (slowing down cores) and dark-silicon (keeping a majority of the transistors off or utilizing them for routing) [2].

It is evident that further performance gains would need to bind the application and hardware closer using application-specific integrated circuits (ASIC). A move towards this, while preserving the general-purpose programmable hardware, is the growing dependence on specialized [3] on-chip hardware in the form of graphics processing units (GPU), field programmable gate arrays (FPGA), and digital signal processors (DSP). The host CPU would offload suitable computational tasks to connected accelerators based on the application and structure of the program. Different accelerators, as shown in Figure 1.1, are chosen based on their strengths that enable better execution of a particular type of logical manipulation. This is an active research area with more specialized accelerators being developed, like neuromorphic chips (mixed-signal architecture reflecting neural structure), ASIC miners (for cryptocurrency mining using blockchain) and tensor processing units (for neural network based machine learning), among others.

Alternate forms of computations are being explored as well. Memristors refer to two-terminal non-volatile memory devices based on resistance switching effects [4]. These are useful for nano-electronic memories, computer logic, and neuromorphic architectures. Another example is DNA computers which use DNA biochemistry instead of silicon-based hardware [5]. It is characterized by slow processing speeds but has po-

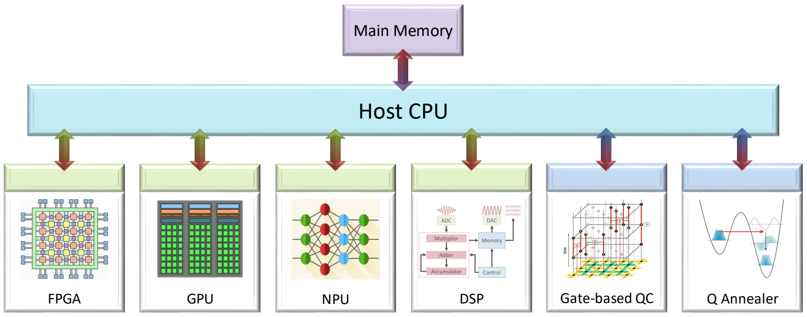


Figure 1.1: The accelerator model, where specialized processors provide high-performance computing power bridging general-purpose processors to application-specific processors. Quantum accelerators (based on quantum logic gates or quantum annealing) are the most viable models of quantum computation towards general-purpose universal quantum computers.

tentially high parallelism. An alternate computing paradigm that is receiving a lot of attention lately in computer architecture is quantum computing - which uses the fundamental properties of quantum mechanics to achieve a computational advantage over classical computation. Within the next decade, it is likely that applications will be a hybrid combination of a classical computer and a quantum accelerator, with multiple computational kernels, instead of a universal quantum processing unit. As a holistic view, we consider two classes of quantum accelerator [6] as additional co-processors. One is based on quantum gates, and the second is based on quantum annealing. The classical host processor arbitrates the control over the total system and delegates the execution of certain parts to the available accelerators.

1.2. QUANTUM COMPUTERS

At the turn of the twentieth century, the fundamental laws of physics were consolidated under just two (albeit not fully compatible) pillars: general relativity and quantum mechanics. Over the course of the following decades, these theoretical principles of the first quantum revolution were applied in various technologies like lasers, satellite-based positioning, magnetic resonance imaging, and transistors. The core of these advancements relied on phenomena that manifest in the microscopic scale of atomic and sub-atomic particles and thus have remained counter-intuitive to natural human experience.

These quantum phenomena, specifically superposition, entanglement, interference, and tunneling, are best formulated by extending probability theory to the complex number domain. This led to the development of the field of quantum information theory. The idea of using these fundamental physical building blocks of nature for computation [7] laid the foundation for the second quantum revolution, focusing on controlling quantum systems and engineering them for arbitrary computation instead of a passive understanding and harnessing of quantum phenomena. While quantum mechanics deals with examining properties of matter, quantum information science focuses on extracting information from those properties, and quantum computation (QC) manipulates and processes this information via logical operations.

The developments in quantum computation are inspired by both theoretical and engineering perspectives. These primary motivations are presented here.

1.2.1. THEORETICAL MOTIVATION

The earliest venture to explore the domain of QC was purely a theoretical pursuit to understand the computational model and what class of problems are computable and efficiently solved by it. The origin of this idea is in the intrinsic parallelism offered by the superposition of quantum states. This allows manipulating the superposed quantum wave function by a single logical quantum operator. However, the retrieval of the output of the computation involves a measurement process that irreversibly destroys the superposition and probabilistically generates a single quantum state as the observable. Thus, the quantum algorithm cleverly guides the interaction among the superposition states via constructive and destructive interference to increase the probabilities of the states that encode the answer from the intended logical manipulation.

Computability theory studies the problems that are solvable for a specific computing model. Surprisingly, quantum computation does not propose a computing model more powerful than classical computers (Turing machines). Thus, in principle, all mathematical functions that are computable using a QC are also computable classically. This is very advantageous in these initial days of QC research as it allows simulating quantum computation on classical hardware, i.e., quantum computing simulators.

The promising advantage of QC is rather manifested in the computational resources needed to complete the task. The most studied of these are runtime (time) and memory (space). Complexity theory bridges the gap between practical algorithms running on computing hardware made out of simpler grammar of programming languages and the hierarchy of recursive languages in computability theory. The complexity of algorithms has been classified into a multitude of classes. The boundaries and relationships between these classes are sometimes not proven but are based on current knowledge and popular beliefs in the scientific community. The complexity classes of P and NP, and their relation to quantum complexity classes are of immediate interest for efficient algorithm development.

To compare the efficiency of quantum algorithms, it is crucial to compare their complexity with the current best algorithm on classical computers. Bounded-error quantum polynomial (BQP) time is the class of decision problems solvable by a quantum circuit whose length and the number of qubits scale polynomially with respect to the instance size. Like the classical complexity class of BPP (bounded-error probabilistic polynomial) time, the error probability is bounded by at most 1/3 for all instances; BPP being the class considered to be practical on a classical computer. Using Chernoff bound, the constant 1/3rd can be reduced arbitrarily on repetition. As P is a subset of BQP, it is interesting to study algorithms that fall outside P but in BQP. Such algorithms include integer factorization (Shor's algorithm), discrete logarithm, Jones polynomial approximations for certain roots of unity, etc. Thus, though quantum computers might not be able to make NP-complete problems efficient [8], there are problems of interest in BQP as well that quantum computers can solve efficiently.

It is interesting to note that for many problems in NP, quantum algorithms offer polynomial speedup, which can boost these problems into the domain of practicality (like

DNA sequence reconstruction, as explored in this dissertation). Another recent direction in quantum algorithms is to design novel heuristics without rigorous complexity theoretic proofs of optimality. These heuristics might be promising candidates for a quantum computational advantage as it is not immediately clear how to translate these to corresponding efficient classical computation. Based on current physical theory, quantum computers are the most general kind of computers physically allowed [9]. More exotic computers would require some refinement of the laws of physics (like non-linearity allowing faster than light travel, violating the uncertainty principle, or using closed timelike curves).

1.2.2. ENGINEERING MOTIVATION

Gordon Moore observed [10] in 1965 that the number of components per integrated circuit doubles approximately every two years. Moore's law has proved accurate for more than four decades and has been instrumental in guiding the research plans and targets for the semiconductor industry. However, it is an observation or projection based on human ingenuity, not a physical or natural law. This trend has endured the test of time through several enabling factors like complementary metal-oxide-semiconductor (CMOS), dynamic random-access memory (DRAM), chemically-amplified photoresist, deep UV excimer laser photolithography, etc. Chip manufacturers like TSMC, Intel, and IMEC [11] expect to shift from GAA FinFET to nanosheet and CFET to push the limit to 0.7 nm process around 2030. Nature being inherently quantized, classical computation adheres to the laws of quantum mechanics as well. The quantum-computing-specific phenomena however act in the atomic/sub-atomic realm. Moore's transistor scaling law is fast approaching this size where quantum effects will be unavoidable. In quantum computers, these phenomena are harnessed for useful computation.

One of the first explorations towards harnessing quantum effects for computation was aimed at reducing the energy consumption of processors. Landauer's principle provides the theoretical lower limit of energy consumption of computation. It holds that any logically irreversible manipulation of information, such as the erasure of a bit or the merging of two computation paths, must be accompanied by a corresponding entropy increase in non-information-bearing degrees of freedom of the information-processing apparatus or its environment, i.e., by heat generation. Quantum logical manipulations are reversible unitary transformations; thus, theoretically can work without spending energy (except during the last step of measurement). However, in practice, running current QC requires a lot of energy for dilution refrigerators used for cooling the processors to near absolute zero Kelvin, to reduce thermodynamic noise. This connects to research in room-temperature superconductors which might allow quantum computers to break the frontiers of green computing.

Current QC efforts are focused on a paradigm shift in high-performance computing instead of energy consumption. Today's efforts in building large-scale qubit processors are targeted toward executing quantum algorithms that can demonstrate the theoretical complexity benefits in runtime, memory, or heuristics. Thus, instead of energy-efficient universal quantum computers, most research is targeted toward quantum accelerators as specialized computing devices in addition to the classical host CPU.

1.2.3. COMPUTING MODELS

Quantum computing is an entire class of computing, and thus, there are a multitude of models corresponding to their classical counterparts. The theoretical models, like the quantum circuit model, adiabatic quantum computing, measurement-based (cluster state) quantum computation, and topological quantum computing are equivalent to each other within polynomial time reductions.

The first proposed model is the quantum Turing machine (QTM) by David Deutsch [12], corresponding to the classical theoretical model of the Turing machine. While the QTM model is theoretically simple, it gets complex and clumsy for practical purposes compared to other equivalent models. QTM is typically reserved for studies on computability and quantum automata theory.

One of the most popular and by far the most extensively developed computation models is the circuit model for gate-based quantum computation [13]. This is the conceptual generalization of Boolean logic gates (e.g., AND, OR, NOT, NAND, etc.) used for classical computation. The gate set for the quantum counterpart allows a richer diversity of states on the complex vector space (Hilbert space) formed by qubit registers. The quantum gates, by their unitary property, preserve the 2-norm of the amplitude of the states thereby undergoing a deterministic transformation of the probability distribution over bit strings. The power of quantum computation stems from this exponential state space evolving in superposition while interacting by the interference of the amplitudes. Gate-based quantum algorithms are designed such that the solution states interfere constructively, while the non-solutions interfere destructively, biasing the final probability distribution in favor of reading out the solution(s).

Adiabatic quantum computation (AQC) was conceived specifically to solve satisfiability problems of the NP-complete class. This paradigm is based on the adiabatic theorem, i.e., the possibility of encoding a specific instance of a given decision problem in a particular Hamiltonian. The system is initialized in a ground state of an easy-to-construct Hamiltonian, and slowly (adiabatically), the system is deformed to the desired Hamiltonian where measurement of the final ground state reveals the desired optimal value. The speedup of the algorithm depends crucially on the scaling of the energy gap as the number of degrees of freedom in the system increases. It was shown to be polynomially equivalent to the circuit model, which implies that its application to an intractable computational problem might not be feasible (having intractably many possibilities of getting stuck in eigenvalues of local minima) and remains an open empirical question.

Another common type of quantum system is a quantum annealer. Quantum annealing is connected to the AQC paradigm [14], although there are subtle differences. Unlike AQC, quantum annealers need not be universal for quantum computing, making it relatively easy to engineer a large-scale system. While the circuit model was inspired by Boolean logic circuits, quantum annealing was inspired by the metallurgical process of annealing where by virtue of thermal fluctuations, the material is able to explore more favorable parameters (e.g. crystal size) by thermally jumping over barriers in the parameter space. By imparting heat, the solution parameters can climb local minima. Quantum annealing uses quantum fluctuations instead to tunnel through high but thin barriers in the target function. If the parameter landscape has these specific kind of barriers, it can translate to a computational speedup in finding the minimum (ground state) of the function.

There are many other QC models beyond these most researched ones that were discussed above. Topological quantum field theory (TQFT) model are based on exotic physical systems (topological states of matter). More examples of QC models are Boson sampling and measurement-based quantum computation (one-way computing).

Most QC models use a two-level system, qubit (a quantum binary digit), as the token for quantum information. Like Boolean algebra, this is universal for representing an arbitrary number of levels. Qubits can be generalized to d-level systems, called qudits, and are beneficial for specific algorithms and models of physical implementation.

1.2.4. PHYSICAL IMPLEMENTATIONS

There are various potential candidates for the hardware back-end being pursued as well. Superconducting integrated circuits are Josephson junction based harmonic oscillators, coherently controllable and measurable by magnetic flux pulses and microwaves. Quantum dot architectures are based on individual electrons confined in quantum dots (quantum well potential), controllable using magnetic flux pulses and inter-dot gate voltage, and measurable using tunneling current measurements. Ion traps are based on alkali metal ions confined in a trap using electric fields and controllable/measurable by laser pulses. Other candidates like NMR-type spin qubits, Nitrogen-vacancy center based, photonic qubits, and Majorana Fermion based topological qubits are actively being researched as well.

In summary, there are many different models and design choices that are currently being actively pursued. The theory of universal quantum computation provides the guarantee that an algorithm would preserve its complexity-theoretic efficiency over these hardware choices, allowing quantum algorithm designers to remain agnostic to these developments. However, this guarantee is not applicable for quantum algorithms with polynomial speedup (e.g., Grover search and QAOA), as the polynomial overhead of translation between computing models can negate the speedup with respect to the classical algorithm. Also, many quantum algorithms are oracular, i.e., consider a specific part of the entire algorithm pipeline assuming the remaining can be performed efficiently. The full pipeline of such algorithms might involve steps to implement the oracle that dissolve the speedup of the quantum kernel. Thus, quantum algorithm designers need to be sufficiently aware of the various constraints of the computing hardware and the holistic view of the system pipeline.

1.2.5. QUANTUM INFORMATION AND COMPUTATION

Quantum computation utilizes the laws of quantum mechanics to a computational resource or quality advantage. It is the only realizable model of computation that violates the extended Church-Turing thesis, as classical models (like Turing machines) require a worst-case exponential resource of space or time to simulate quantum computation. Information processing via quantum postulates [15] can be described by:

1. The superposition principle defines the possible states $|\psi\rangle$ of a quantum system. An isolated system is represented as a linear combination of a chosen orthonormal basis states $\{|i\rangle\}$ with complex coefficients $\alpha_i \in \mathbb{C}$, as $|\psi\rangle = \sum_{i=0}^{d^n-1} \alpha_i |i\rangle$. The

complex amplitudes are normalized $\sum |\alpha_i|^2 = 1$, such that $|\psi\rangle$ is a unit vector in an n-dimensional complex vector space or Hilbert space.

2. The evolution of a closed quantum system is described by an operator U typically associated with the Hamiltonian, i.e., $|\psi'\rangle = U|\psi\rangle$. It governs how the state of a quantum system evolves with time. The operator is unitary and thus reversible, $U^\dagger U = UU^\dagger = I$.
3. The measurement principle governs the collapse of the superposition and bounds the amount of accessible information in a quantum state. The Born rule states that the superposition evolves irreversibly to a specific basis state $|i\rangle$ with the probability $|\alpha_i|^2$. Quantum measurements are described by a collection $\{M_m\}$ of measurement operators, where m refers to the measurement outcomes and the probability that result m occurs is given $Pr(m) = \langle\psi| M_m^\dagger M_m |\psi\rangle$. The state of the system after measurement is $M_m|\psi\rangle / Pr(m)$. While measurements in the physical world are qubit interactions (entanglement) between the system and the environment/detector in the joint Hilbert space, it is yet not possible [16] to derive this postulate from the others.
4. The state space of a composite physical system $|\psi\rangle$ is the Kronecker tensor product of the state spaces of the s-component physical systems, i.e. $|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_s\rangle$. Thus the number of parameters needed to describe the state grows exponentially with the number of qubits. This is the primary resource in quantum computation to achieve superior computational capability over classical systems by selectively interfering these states by the algorithm.

The logical abstraction of the basis states is typically associated with a physical meaning. For example, the simplest quantum states are described as a two-level system, i.e. $d = 2$ called qubits, and can be represented by the spin-up/down of an electron.

1.2.6. ACCELERATOR STACK LAYERS

Moving towards a broader view of constructing a functional quantum computer, the requirements from the models must be quantifiable. This is given by the DiVincenzo's criteria [17] for quantum computation and communication (last 2):

1. A scalable physical system with well characterized qubits
2. The ability to initialize the state of the qubits to a simple fiducial state
3. Long relevant decoherence times
4. A universal set of quantum gates
5. A qubit-specific measurement capability
6. The ability to inter-convert stationary and flying qubits
7. The ability to faithfully transmit flying qubits between specified locations

A quantum accelerator is the computation model where a classical processor uses the power of a quantum processor for specific tasks [18]. In the initial days of quantum computers (what is called as the pre-universal quantum computing era), we are most likely to use quantum computers for very specialized applications, where we can use the power of superposition and entanglement for a computational advantage. Thus, even though a quantum algorithm can do everything a classical algorithm can (within polynomial factors), it is not likely that quantum computers will be standalone systems. In our research, we follow the circuit model for gate-based quantum computing on qubits.

The classical processor interacts with the quantum processor via commands that specify a sequence of unitary gate operators and receives the collapsed state of the two-level qubit system on measurement.

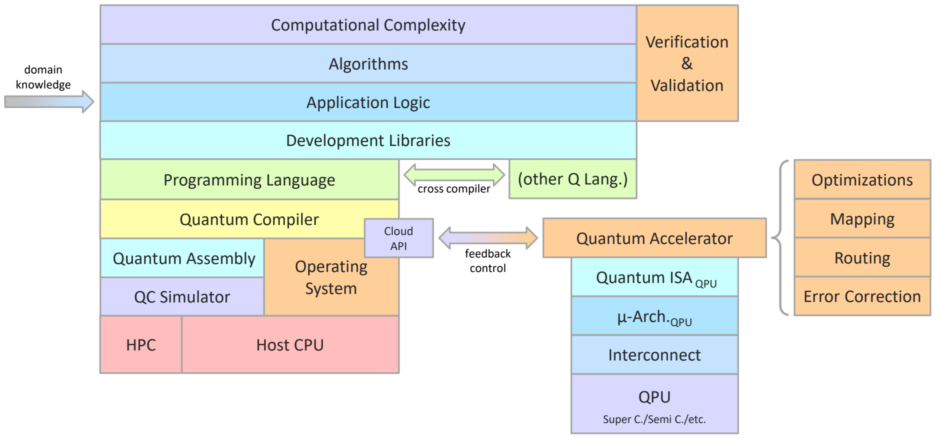


Figure 1.2: The quantum accelerator architectural stack.

The quantum accelerator is architecturally layered; this is called the quantum computing stack [6]. The stack, as shown in Figure 1.2, is inspired by the currently existing classical computers. The major stack layers from the application layer to the physical substrate are:

- The application can be described in hybrid quantum-classical logic as mathematical state evolution designed to perform the desired task. They need to be decomposed into programming constructs as input to the compiler. The logic is based on developed algorithms with associated computational complexities;
- Compilers and programming languages (e.g., OpenQL, Qiskit) are the interfaces for the algorithm designer to precisely define the quantum operators and state in abstracted high-level constructs. To ease the development of algorithms, many compilers now offer libraries that consist of an arsenal of primitives that help a quantum algorithm developer;
- The operating system on the host CPU orchestrates the compiler. The compilation process generates an assembly-level code (e.g., common-QASM, OpenQASM) specifying the gate operations. Hardware agnostics application development (as discussed in this dissertation) directly interfaces the QASM with the simulator (which in turn runs on the classical CPU or a high-performance compute system). The simulated qubits are perfect in nature for testing the functionality of the algorithm.
- Alternatively, a cloud application programming interface connect to a quantum accelerator. Hardware specific optimizations are carried out based on the QPU specifications. The quantum runtime unit is responsible for scheduling the operations. This includes quantum error correction (QEC) and qubit logical to physical mapping. An executable-QASM is generated via this process;

- Quantum Instruction Set Architecture (ISA) defines the runtime operations of both classical control and quantum parts of the algorithm. It encapsulates the hardware dependence. Micro-architecture takes into account the precise timing controls and the instruction pipelines. Quantum-classical interface comprises of ADC and DAC and their controls for interacting with the physical qubits (and cryogenic interface electronics). Finally, the quantum processing unit houses the physical qubits. This can be superconductors, semiconductors, or other types of competing qubit technologies.

1.3. RESEARCH METHODOLOGY

The level of abstraction used for this doctoral research considers the functional algorithm design, demonstrated as a proof of concept (PoC) on a quantum computing simulator. This careful balance between the mathematical description of quantum algorithms and a implementation on available noisy intermediate-scale quantum (NISQ) processors is chosen based on the technology readiness level of various stack layers of the quantum accelerators.

The applications studied in this dissertation start by analyzing the computational bottlenecks and accessing the applicability of a quantum accelerator or quantum information formulation. Thereby, a scalable mathematical model of the quantum solution is formulated. The resource complexity in terms of qubits and gates is then studied to determine at which problem size the application can be demonstrated on both a QC simulator and available NISQ processors. If the problem size is found to preserve the value as a PoC, i.e., it is possible to demonstrate the quantum advantage, a quantum program is coded. The results from this entire application pipeline are tested and analyzed.

Most applications discussed in this research are not targeted to be the first demonstration of a quantum advantage experimentally. Instead, the focus is on high-impact applications that possibly would need much larger quantum processors for implementation, but have stronger guarantees to be suitable for formulations based on quantum information.

1.3.1. FUNCTIONAL ALGORITHM DESIGN

Quantum algorithm design translates the requirements of the application on quantum programming languages. Thus, it interfaces with various target applications and the target quantum compilers.

For a holistic quantum accelerated application pipeline, implementing a PoC of the quantum kernel is not sufficient. The developed PoC should take into account any classical pre-processing and post-processing steps, as well as the complexity of compilation and real-time control during the algorithm execution. Though a quantum algorithm designer might not address improvements on the entire pipeline, the assumptions on the auxiliary parts should be clearly understood to assess the overall impact of the research. In principle, most of these parts are assumed to be offloaded to the underlying stack layers like compiler code optimizer and runtime micro-architecture. An isolated discussion of a specific quantum algorithm is insufficient for near-term implementation.

The quantum algorithm would have interfaces with other software modules running

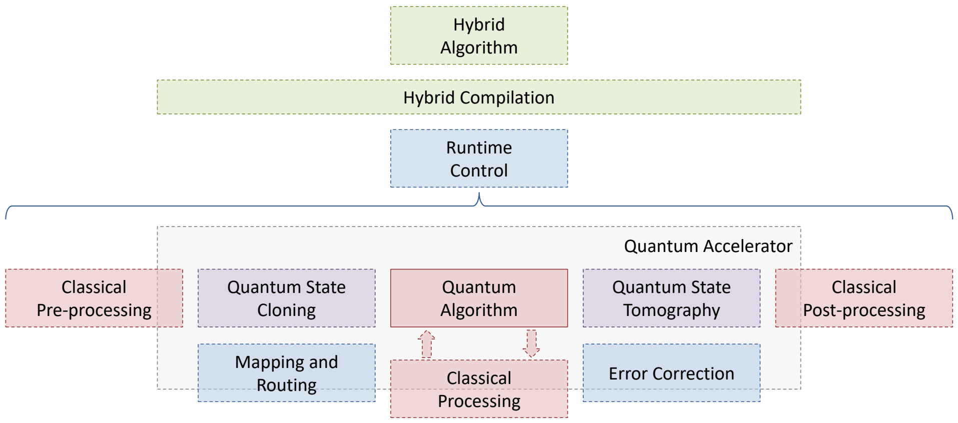


Figure 1.3: Quantum Algorithm (block with solid outline) and interfacing blocks within a quantum accelerator that influence the overall speedup.

in parallel, as shown in Figure 1.3. The application logic is described as a hybrid algorithm (green block in the figure), i.e., a classical program with quantum kernels. The hybrid compilation identifies the parts that are executed on the quantum accelerator. During the execution of the quantum algorithm, the quantum algorithm interfaces with other blocks within and outside the accelerator. These can be functionally grouped under three factors that contribute to the overall runtime of a general quantum algorithm:

- Application logic (red blocks in the figure): This pertains to the core algorithm running on a simulator, where the internal state vector can be accessed. It refers to the inherent gate complexity of the algorithm and other classical pre/post-processing involved;
- No-cloning (violet blocks in the figure): If the internal state vector cannot be accessed (like in real quantum processors), the experiment needs to be repeated multiple times, and the measurement is aggregated. Most algorithms demand a statistical estimate of the state's probability distribution. The central tendency of these measurements is the resultant output from the quantum algorithm. Quantum state tomography is an active area of research. Similar constraints are present in the classical data input in the absence of a QRAM.
- Experimental (blue blocks in the figure): For algorithm development (using perfect qubits) and PoC testing, a simulator platform is preferable, such as the QX Simulator used for this research. After sufficient confidence in the logic is established, it needs to be ported to an experimental quantum processing unit [19]. This adds complexity overhead for topological mapping and routing [20], quantum error correction cycles [21] and runtime control latencies.

Thus, every quantum algorithm that depends on a probabilistic result in a noisy environment needs to be repeated, adding a multiplicative factor to the inherent gate complexity.

There are conceptual differences between quantum accelerators and parallelization provided by other accelerators like GPUs. The data input-output bottleneck between classical and quantum makes quantum suitable for 'small data, big compute' instead of big data. Our research on quantum accelerated algorithms focuses on finding niche applications that can benefit from this software model.

1.3.2. PROGRAMMING FRAMEWORKS

In this doctoral research, multiple quantum programming frameworks were used - OpenQL, QX Simulator, Qiskit and Ocean. These choices were initially based on ease of access to tools developed within the research group that allowed easy debugging and feature requests. Eventually, the implementation platform was shifted to emerging de facto standards in the quantum computing community.

The OpenQL framework allows hybrid quantum-classical coding in Python or C++, compiling and optimizing quantum code to produce the intermediate Common QASM (cQASM) [22] and the compiled Executable QASM (eQASM) for various target platforms (superconducting qubits, spin-qubits, NV-centers, etc.) The Qxelarator library allows the execution of the compiled QASM on the QX binary and receives the measurement outcomes in the high-level OpenQL code encapsulating the quantum architecture (in this case, a simulator), allowing interleaving classical and quantum code blocks in a single program. QX is a universal quantum computer simulator that takes as input a cQASM file and provides thorough aggressive optimization, high simulation speeds for qubit state evolution. The experimental setup used (with 28 HT cores at 2.00 GHz and 384 GB memory) can simulate ≈35 qubits if the states and operations are non-sparse. Qiskit is the open-sourced quantum programming framework from IBM. It allows easy integration with the IBM qubit processors. Over the course of the research, Qiskit has become popular in the quantum education and development community, which prompted its adoption for the later implementations in this work. The application of de novo DNA sequence assembly were also implemented on a quantum annealer. The Ocean framework from D-Wave was used for executing it on the annealing simulator and the D-Wave quantum annealer.

While libraries (like OpenQL and Qiskit) in high-level languages allow integrating quantum programming with classical logic, the description of the quantum logic is explicitly specified at the circuit level. Most of these programming models use function calls to modify the quantum program object by appending specific unitary gates at the specific qubits. Thus, conceptually the level of abstraction is the same as a quantum assembly (QASM) language. However, the high-level language features like variables, loops, and conditional statements allow for a more compact representation compared to QASM.

1.3.3. PERFECT SIMULATION

The period of this doctoral research has been witness to various landmark events in the quantum computing timeline. These include demonstrations of quantum supremacy, a shift to NISQ algorithms, and various record-breaking funding and research efforts. Despite these, the justification behind the chosen research methodology has remained valid.

The shift in focus from large-scale error-corrected fault-tolerant quantum processors to NISQ era has been triggered by John Preskill's talk [23]. The enabling factor of this is the availability of cloud-based quantum processors on a few qubits from vendors like IBM, Rigetti, IonQ, Quantum Inspire, etc. However, there remain considerable challenges in various factors like qubit multiplicity, gate error rates, coherence time, connectivity, and high-level programming features for these devices to be applicable as PoC for the applications discussed in this research. Thus, we remain agnostic to the availability of these hardware platforms as they do not meet the technology readiness level required for practical applications. The algorithms are developed on perfect intermediate-scale quantum (PISQ) computing simulators as an algorithmic level abstraction of future FTQC (fault-tolerant quantum computing), as shown in Figure 1.4. Taking into account the layers of quantum error correction, mapping and routing, even within quantum simulators, shifts the application from being demonstrable to intractable. Moreover, these do not add to the quantum logic for the algorithm, as future quantum compilers are expected to abstract these from the quantum programmer. Thus, these are not considered in this research.

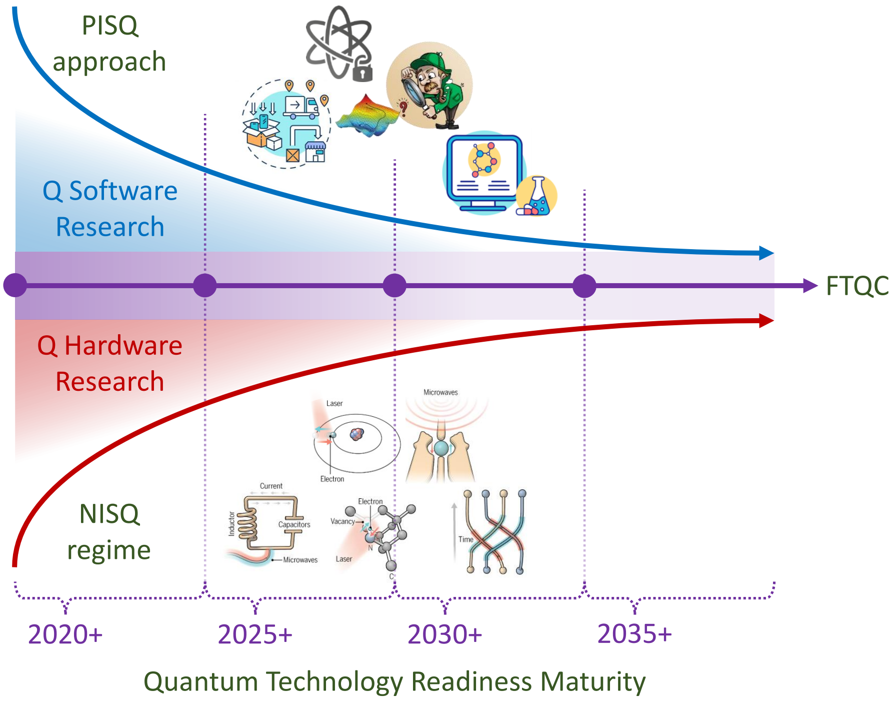


Figure 1.4: Technology readiness gap between quantum hardware and quantum software readiness. It is currently not favorable to develop proof of concept demonstrations via hardware-software co-design. For this research, we follow a Perfect Intermediate-Scale Quantum (PISQ) computing simulation approach.

A breakthrough has been various demonstrations of quantum supremacy [24], where a quantum processor is made to execute a specific circuit that is provably inefficient (often requires intractable amounts of runtime, in multiple years) on supercomputers. However, these algorithms cleverly exploit the strengths of quantum computation and are, in principle, not usable for any real world application. In this research, we focus on quantum advantage, where the quantum formulation provides computational benefit to an application that has a societal, industrial, or scientific impact.

The algorithms developed in this research are tested on noise-free, fully connected quantum computing simulators. Simulation limits based on the exponential state space scaling are around 35-50 qubits (between personal computers and supercomputers). Yet, this paradigm of perfect intermediate-scale quantum computing simulation has been beneficial for demonstrating the proof-of-concept of the quantum approach at the current technology readiness level.

1.4. DISSERTATION ORGANIZATION

The remaining chapters of the dissertation describe three specific use cases studied in this doctoral research:

- In § 2, we study the use case of DNA sequence reconstruction. Quantum formulations for two different applications for this use case, ab initio reference alignment and de novo read assembly, are discussed.
- In § 3, we focus on the circuit formulation of quantum automata models and their encoding for studying mathematical objects in algorithmic information theory. This framework is applied for genome analysis and causal modeling use cases.
- In § 4, we formulate a universal reinforcement learning agent framework that can model quantum environments, test foundational principles, and can be used for optimizing quantum algorithms.

In § 5, we conclude the dissertation with suggestions on future directions.

The use cases discussed in the various chapters are fairly independent of each other, such that these chapters can be read separately. Each application requires a different set of backgrounds based on their domain of applicability, which is introduced independently. The gradual development of the reasoning behind the consideration for each use case and how it connects to the research methodology presented in this chapter can be appreciated from the concluding remarks in each chapter.

2 Quantum-accelerated genome sequence reconstruction

The research question formulated at the inception of this doctoral research is to understand use cases of the quantum computing stack being developed by the Quantum Computing Architecture team in collaboration with QuTech. This involved investigating candidates for long-term high-impact applications of quantum computing. The requirements for the stack layers, like compilers, micro-architecture, and QC simulator, are typically driven by the requirements for qualifying quantum processing units, e.g., randomized benchmarking and surface-code QEC cycle. The aim of this research is to create an alternative, top-down, application-driven vision of the requirements for the quantum accelerator stack. Thus, being synchronized with the current state (size and quality) of quantum computing hardware is not the primary focus. Instead, the algorithm design and theoretical study of the resource scaling would inform the tipping point of quantum advantage for the target application.

The supervisory team of this doctoral research have considerable experience in the domain of high-performance genomics accelerators. This is in the context of the startup company, Bluebee (acquired by Illumina in 2020) that offers cloud-based genome analytics on FPGAs. Thus, the first task undertaken was to investigate possible quantum acceleration candidates within the genomics HPC pipeline to further empower this application of high societal benefit. Given the high demand for processing power, quantum acceleration for the specific step of genome sequence reconstruction is explored

This chapter is based on the following:

- **Sarkar, A.**, Al-Ars, Z., Almudever, C. G., & Bertels, K. L. (2021). QiBAM: Approximate Sub-String Index Search on Quantum Accelerators Applied to DNA Read Alignment. *Electronics, 10*(19), 2433.
- Krol, A. M., **Sarkar, A.**, Ashraf, I., Al-Ars, Z., & Bertels, K. (2022). Efficient decomposition of unitary matrices in quantum circuit compilers. *Applied Sciences, 12*(2), 759.
- **Sarkar, A.**, Al-Ars, Z., & Bertels, K. (2021). QuASeR: Quantum Accelerated de novo DNA sequence reconstruction. *Plos one, 16*(4), e0249850.

in this chapter. The two solutions developed as part of this investigation of quantum-accelerated DNA sequence reconstruction are presented.

2.1. USE CASE MOTIVATION

Understanding the genome of an organism reveals insights [25] with scientific and clinical significance, like causes that drive cancer progression, intra-genomic processes influencing evolution, enhancing food quality and quantity from plants and animals. Genomics data is projected to become the largest producer of big data within the decade [26], eclipsing all other sources of information generation, including astronomical as well as social data. At the same time, genomics is expected to become an integral part of our daily lives, providing insight and control over many of the processes within our bodies and in our environment. An exciting prospect is personalized medicine [27], in which accurate diagnostics can identify patients who can benefit from precisely targeted therapies.

Despite the continual development of tools to process genomic data, current approaches are yet to meet the requirements for large-scale clinical genomics. In this case, patient turnaround time, ease of use, robustness, and running costs are critical. As the cost of whole-genome sequencing (WGS) continues to drop [28], more and more data is churned out, creating a staggering computational demand. Therefore, efficient and cost-effective computational solutions are necessary to allow society to benefit from the potential positive impact of genomics. This research provides efficient solutions based on the quantum computing paradigm to the high computational demands in the field of genomics, specifically for genome sequence reconstruction.

2.2. GENOME SEQUENCE RECONSTRUCTION

Nucleic acids like DNA and RNA are very long, thread-like polymers made up of a linear array of monomers called nucleotides. These carry the genetic instructions used in the growth, development, functioning, and reproduction of biological organisms. These genetic instructions are primarily encoded in the sequence using the four nucleic molecules, adenine (A), cytosine (C), guanine (G), and thymine (T). Adenine pairs with thymine, and guanine pairs with cytosine, represented by A—T and G—C, which are referred to as base pairs (bp) in the DNA. The information in the two strands of the DNA is thus complementary. This allows simplifying the representation of both DNA and RNA as a single string with four distinct symbols while processing it as digital data.

The length of genomes varies greatly among organisms, for example, the human genome is approximately 3.289×10^9 bp long. Owing to this length, it is not possible to obtain the entire sequence in a single readout from the sequencing machines.

In order to sequence the organism, multiple copies of the DNA/RNA are broken down into fragments as sequencing machines are not capable of reading the entire genome at once. Then, these fragments are sequenced using modern sequencing technologies (such as Illumina), which produces reads of approximately 50 to 150 bp at a time, with some known error rate. Then these short strings are stitched back together - a process called sequence reconstruction.

Genome sequence reconstruction is primarily done using two techniques:

1. de novo assembly of reads
2. ab initio alignment of reads on reference

De novo assembly is applied while sequencing a new organism, where the short reads are stitched back together based on the overlap between each pair. This is computationally very intensive and is impractical for classical high-performance computing except for small micro-organisms. For organisms with longer genomes, for example, humans, the alignment method is preferred. Once the DNA/RNA is constructed for a species, for example, via the Human Genome Project, this is used as a reference for further individuals of the same species. The whole genome is reconstructed by aligning the short reads on the reference genome. Thereafter, the variation from the reference genome can be inferred to understand specific traits or abnormalities in the individual.

Since the principles of quantum computation are fundamentally different, we investigate the most basic algorithmic primitive for which the quantum kernel can be constructed. § 2.3 presents quantum-accelerated read alignment, while in § 2.4 quantum-accelerated read assembly is presented.

2.3. QUANTUM-ACCELERATED READ ALIGNMENT

Some of the most promising applications for quantum acceleration are physical system simulation, cryptography, optimization, and machine learning. QiBAM (quantum indexed bidirectional associative memory), the technique developed for this application, falls under the umbrella of quantum search-based algorithms, where a high dimensional state space of qubits is harnessed to explore/search an optimization landscape faster and better. While these generic algorithms are ubiquitous in computer science and data structures, in this research, an exemplary case of DNA sequencing application is considered in depth. This is motivated by the immense application of this area in the upcoming years and its reliance on a high volume and speed of data processing. Faster DNA sequence reconstruction will assist the adoption of precision medication by accelerating the diagnostics pipeline.

Bioinformatics algorithms in use today (especially the focus here, that is, DNA sequence alignment) rely on heuristic methods to cope with the huge volume of data. Even these heuristic approaches take days to run on supercomputing clusters, limiting their applicability to wider use. The advantage of quantum algorithms discussed in this work is twofold: (a) they have a lower runtime than corresponding classical algorithms; and (b) a global optima is guaranteed to be sampled with the highest probability instead of a sub-optima as in current classical heuristics. Application areas like cryptography and physical simulations are highly susceptible to noisy input/computation, which makes them highly improbable to achieve good results in the low coherence quantum systems that will be available in the near-term. The application for DNA alignment searches for a sub-optima within an acceptable threshold, so an approximate solution is more permissive for this use case.

Quantum approaches to DNA sequencing have not been explored in much depth before. The master's thesis research as part of this doctoral research's scoping is the first time [29] a gate-based quantum algorithm has been presented where the DNA index of the best matching sub-string is retrieved with high probability. While previous work on associative memory and phone directories provide the tools for this application, in this research, a holistic gate-level description of the entire algorithm and the oracle is provided. The specific design in the context of genome sequence reconstruction takes into account the following requirements:

- in genomic sequences, since reads can contain errors, an approximate matching query is required;
- a constant oracle is useful as compiling the oracle differently for every read at run-time is tedious;
- the associated index in the reference needs to be retrieved instead of the corrected query.

The proposed approach, which takes into account these requirements, is tested and verified on a quantum simulator with small artificial sequences as a proof of concept.

2.3.1. SEQUENCE RECONSTRUCTION USING READ ALIGNMENT

Starting as an abstract algorithm, the problem is gradually formulated towards the intended application in genomics. A database is defined with indices and an associated data element for each index, as illustrated with a simple example in Figure 2.1. A search query on this database is provided. It is possible that the exact match for the element is not present in the database. The objective of an approximate index search is to return the index of the element that is closest to the search query. In this example, by visual inspection, it can easily be inferred that the closest match is the element at index 1. In fact, the key cut pattern on both the search query and the nearest matching keys are the same, a metaphor for the usefulness of the nearest association being of functional use.

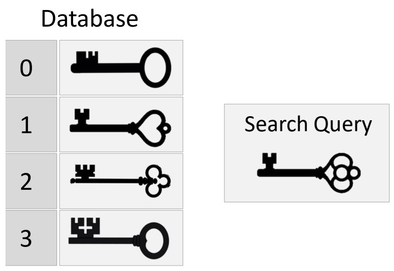


Figure 2.1: Simple example of associative search.

The model of associative memory is closely related to how learning occurs in the brain, thus finding its use in computational neuroscience. It is useful when we have noisy or incomplete knowledge of the data. An indexed memory variant is useful when we not only want to recover the nearest matching data from the database but also the index of its occurrence that has associated semantic meaning. For example, an object detection algorithm tries to find the nearest match to objects (e.g., humans, cars, traffic lights) in a scene. After the nearest match is detected, the index of the match location can help in positioning the object in the scene, for example, if a human is on the left or right of the road. The scene here is the database, and the output is the pixel coordinate of the detected object, based on which important decisions might be automated in a self-driving car.

Here we focus on one such application of approximate index search, where the data are one dimensional (thus, a sub-string search). These data are DNA sequences where finding the index of the nearest match to a query is of immense computational value in bioinformatics.

The Broad Institute's GATK DNA processing pipeline [30] is a widely used toolset for this purpose, which includes several processing stages like map-to-reference, duplicate marking, and variant calling. One of the most compute-intensive processing stages is the map-to-reference stage used for aligning the reads for reference-based DNA reconstruction [31]. Due to the huge data volume of over-sampled reads, whole-genome sequencing (WGS) of a single human take days on computing clusters, limiting the applicability of WGS in personalized medicine. This motivates the demand for acceleration using quantum computation, as even a polynomial speedup can provide huge benefits on a production environment.

Techniques currently used in the genomics industry depend heavily on heuristics to tackle the volume of data that needs to be processed. However, the heuristics used in industrial alignment algorithms, for example, BWA-MEM, are trade-offs between the quality of the solution and the tractability on the computing platform. Given access to a superior computing paradigm, the sequence reconstruction algorithm is thus built bottom-up to achieve the best possible quality. In this research, a heuristic-free quantum algorithm primitive is constructed to achieve a high-performance global alignment algorithm. The quantum algorithm corresponding to a naive sub-sequence alignment is presented, which can currently be implemented as a proof of concept using quantum computing simulators. The presented algorithm can further be refined in the future by adding a gap penalty and dataset-specific heuristics when quantum processing platforms mature to the stage where these algorithms can be implemented in a quantum accelerator.

In order to map a sub-sequence of characters (or short read) to a reference sequence, the Levenshtein edit distance is commonly used as a metric for approximate matching of the sub-sequence, spanning the comparison length. The Levenshtein edit distance is upper bounded by the Hamming distance between the two sequences. In this work, the Hamming distance is used as the cost function for matching, owing to its simplicity for the quantum implementation. Given the reference sequence T and a short read P of length N and M, respectively, the sub-sequence alignment problem is defined as the index $i \in N$ of T, where the alignment of P starts, which gives the minimum edit distance. The short read is matched for each of the $N - M + 1$ starting indices in the reference genome. The alignment algorithm outputs the index of the minimum Hamming distance and optionally, the nearest match in the reference. Note that this concerns the typical problem setting of linear nuclear DNA instead of circular organellar DNA like in mitochondria or chloroplasts.

An example of this naive alignment approach is illustrated in Figure 2.2. The four colors represent the four bases in the DNA, which can be encoded as a four-level system (radix-4 number) or with 2 bits (or qubits) each. The short read, in this case, can be aligned at $32 - 5 + 1 = 28$ locations on the reference genome, resulting in a Hamming distance for each match. For example, at alignment index 0, only the green (second base of the short read) matches with the reference, thus, resulting in a Hamming distance of

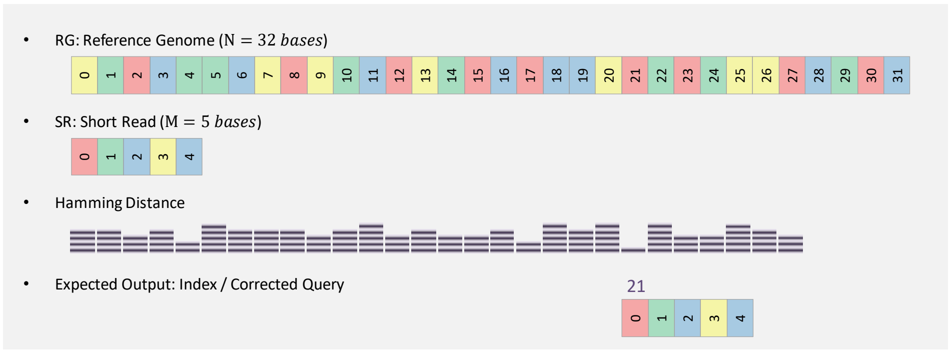


Figure 2.2: DNA sub-sequence alignment problem.

4. By running a classical linear search, it can be inferred that the minimum Hamming distance (of 1) occurs at the index 21. It is important to note that, although the reference genome and short read are of the same species (e.g., humans), an exact match might not be found. This variation can be from two sources, read errors in the sequencing machine, as well as, genetic variation from individual traits or abnormalities with respect to the reference. Thus, it is important to account for approximate matching in the quantum algorithm design.

The core idea behind classical algorithms that improve on the naive approach is developing a strategy such that, after a mismatch, shifts more than one place in the pattern comparison position. The performances of exact string matching algorithms like Boyer-Moore and Knuth-Pratt-Morris [32] degrades quickly as they are not suitable for an approximate match which is very common in DNA sequences due to read errors. The minimum alignment cost for Global Sequence Alignment can be found by the Needleman-Wunsch algorithm, or the Local Sequence Alignment maximized over all possible substrings by the Smith-Waterman algorithm. The industrial approaches (like BWA-MEM) are heuristics built on these basic algorithms to reduce cost in terms of memory and processing while trading off accuracy.

There are many different algorithms for alignment with no single universal preferred method, as different heuristics work for different sequences and applications. Current techniques trade speed and memory for accuracy. These heuristics make the problem tractable for higher-sized genomes like that of a human. However, the approximations and errors introduced prevent further progress in critical application domains like personalized medicine. Thus, in this research, we explore quantum acceleration of the heuristic-free naive approach.

2.3.2. QUANTUM SEARCH

This research modifies the quantum search algorithm [33, 34] on an unstructured database, as proposed by Lov Grover. Grover's search offers a quadratic speedup over a classical linear search. A quadratic speedup may seem less lucrative with respect to exemplary quantum algorithms (such as Shor's factorization); however, it is the only search method possible for unstructured data and is provably optimal [35] in query complexity. Thus, under reasonable assumptions of computational complexity classes (e.g., $P! = NP$), Grover's search based approach is the best algorithm in both classical and quantum domains. Near-term approaches based on Quantum Approximate Optimization Algorithm (QAOA) are now being developed to bridge the gap between Grover's search and current hardware limitations. The speedup from such quantum heuristics are yet to be theoretically proven and are heavily dependent on the hardware specifications (noise characteristics, connectivity, ansatz, gate-set, etc.). Since DNA sequence reconstruction requires more advanced quantum hardware, a hardware agnostic approach is employed, focusing on a coherent protocol that preserves the speedup from quantum search. It is unlikely that DNA sequence reconstruction will lead to a higher speedup with our current understanding of the encoding structure in the data. However, a polynomial speedup can prove to be path-breaking in industrial pipelines, where improvements by state-of-the-art alignment heuristics mostly progress by constant factor speedups for specific datasets.

Grover's search consists of three main steps between state initialization and measurement, as shown as the quantum circuit blocks in Figure 2.3. The algorithm creates a uniform superposition of all basis states by applying the Hadamard gate on all qubits in the all-zero state. The black box oracle marks the solution state. Then the amplitude of the solution state is amplified by an inversion-about-mean operation by the Grover diffusion gate. Repeating the last two steps a quadratic number of iterations with respect to the number of qubits, leads to a high probability of measuring the marked solution state. Thus, the search reduces the query complexity to the oracle by a quadratic factor compared to a classical linear search.

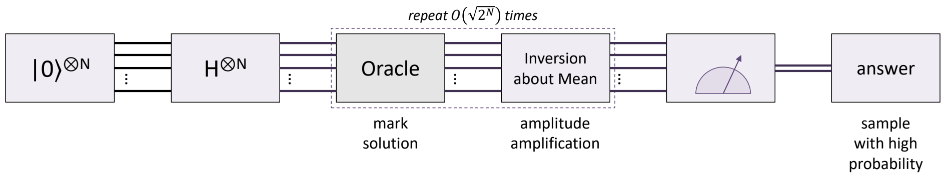


Figure 2.3: Grover's search steps.

Grover's search was enhanced by subsequent research that will allow us to apply this algorithm in our context. These improvements are:

- multiple known number of solutions [36];
- arbitrary distribution of initial amplitude [37];
- multiple unknown number of solutions by randomizing iterations over multiple runs [38];
- multiple unknown number of solutions by a priori counting the number of solutions [39].

The encoding of an application [40, 41] to the oracle is not explicitly described in these papers, and Grover's original paper assumes the execution of the oracle in constant time, for an overall polynomial speedup. However, a complete description of an algorithm in the circuit model needs an explicit representation of the construction of the oracle with quantum gates as described in this work.

2.3.3. RELATED ALGORITHMS

Associative memory, also called content-addressable memory (CAM), is a type of memory organization where instead of the index of the element to be retrieved (similar to a random-access memory, RAM), a partial description of an element is passed as the input query. The element in the memory with the nearest match to the query is retrieved. The differences and application of these approaches are reviewed here, without the detailed proofs of the quantum circuit construction from the original articles.

QUANTUM ASSOCIATIVE MEMORY

The idea of quantum associative memory (QuAM) [42–46] was developed under the umbrella of quantum neural networks (QNN). Intuitively, the entire parallel search operation is reduced to operations on a superposition of states (memories). This results in either an exponential increase in the capacity of the memory, or a reduction in the number of comparisons to constant time.

The algorithm consists of two major blocks, a pattern store and a pattern recall. The pattern store starts from an all-zero initial state as is standard in all gate-based quantum algorithms. The information of the reference text string or DNA, T, is encoded as a superposition database of smaller sub-strings. This set T_M of length M has sub-strings $T_M(i)$ (where $i \in \{0 \dots (N-M+1)\}$) made from T, each starting from a consecutive index. This is compared with a recall pattern, P representing the query. If an exact match in the stored database is found, it is retrieved as the measurement output. However, if a partial or approximate version of P is queried, that is, some characters of the string are not known exactly, the algorithm returns a random output (not correlated with the degree of closeness). Since in practice, P can be inaccurate, our algorithm should be able to retrieve the most similar string from the stored database.

QUANTUM ASSOCIATIVE SEARCH

A major improvement for the quantum associative memory is the use of distributed queries [47]. Using this concept, the associative memory solves the pattern completion problem; that is, it can restore the full pattern when initially presented with a partial pattern such that the known parts exactly coincide with some part of a valid full pattern. This allows the associative memory to also retrieve valid memory items when presented with noisy versions of a partial pattern. This improvement solves the problem of associative search for which no part of the input stimulus is guaranteed to be noise-free. It is desirable to retrieve the memory state, which is most similar to the given stimulus. This kind of memory is called a pattern correcting associative memory.

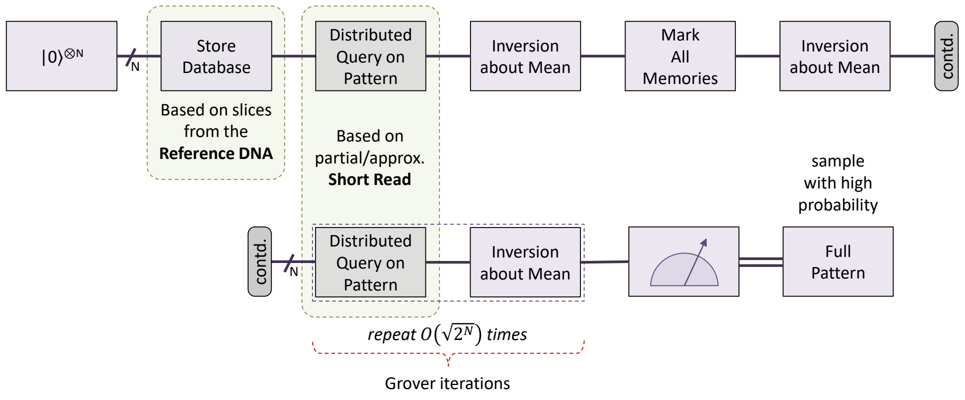
For strings containing only 0 s and 1 s (binary alphabet), this corresponds to finding the minimum Hamming distance between the query and the memory states. Amplitudes are distributed in the distributed query such that the maximal value occurs for some definite state p (the provided search query pattern) and the amplitudes of the other states x decrease monotonically with Hamming distance $h(p,x)$. The binomial distribution matches the required query model. p is the query center of the binomial distribution. Let $d = |x|$ be the number of qubits required to store the memory states.

For all $x \in \{0\ldots(2^d-1)\}$, let

$$|b_p^x\rangle = \sqrt{\gamma^{h(p,x)}(1-\gamma)^{d-h(p,x)}}$$

where γ incorporates a metric into the model, which tunes the width of the distribution permitting the comparison of the similarity of the stimulus and the retrieved memory at a variable scale. The unitary oracle transformation can be formed as

$$O = I_{2^d \times 2^d} - 2\,|b_p\rangle\langle b_p|$$



$|0\rangle^{\otimes N}$
N
Store Database
Based on slices from the **Reference DNA**
Distributed Query on Pattern
Based on partial/approx. **Short Read**
Inversion about Mean
Mark All Memories
Inversion about Mean
contd.
sample with high probability
contd.
N
Distributed Query on Pattern
Inversion about Mean
Full Pattern
repeat $O(\sqrt{2^N})$ times
Grover iterations

Figure 2.4: Quantum associative search algorithm with distributed query.

Further modification [48] to the model of a distributed query is carried out by merging the concept of the memory state oracle with the binomial function based oracle. This is depicted in Figure 2.4. After the pattern is marked by the binomial oracle, the entire superposition for stored memories are marked and amplified. Thereafter, the standard Grover iteration is carried out. The reconstructed pattern from among the stored memories (entries in the database) is retrieved with high probability once the qubits are measured. While the search function is similar to our use-case, the oracle in this case needs to change for each short read and the reads needs to be indexed with respect to the reference.

QUANTUM INDEXED MEMORY

The location in the reference database of an exact or closest match to a query pattern is the alignment index with the minimum Hamming distance between the sequences. This method [49] was developed for a similar problem of amino-acid sequence matching. A block diagram of the algorithm is shown in Figure 2.5.

The initial state is composed of two quantum registers of $N-M$ and M qubits; the index and the pattern forming the quantum phone directory (QPD), similar in architecture to a phone directory with name and number. Initially, the index is set to a full superposition. Then, based on the index, the data is stored in the database. For each tagged index,

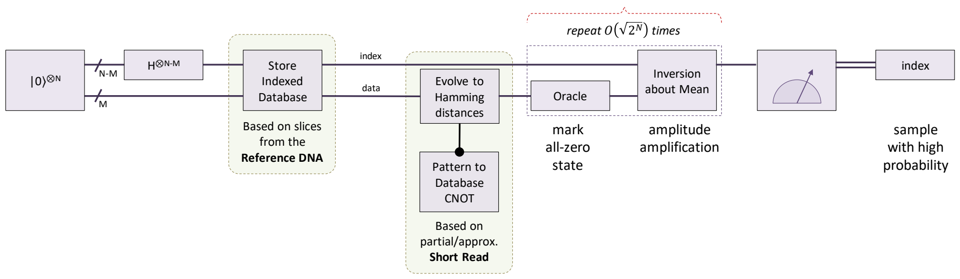


Figure 2.5: Quantum phone directory algorithm.

a sub-sequence of the reference DNA is stored as a basis state of the quantum superposition. Essentially, the set of patterns are sorted into an ordered list due to the second register of the database that tags the data. The initial state is described as:

$$|\psi_0\rangle = \frac{1}{\sqrt{N-M+1}} \sum_{i=0}^{N-M} \Big(|T_M(i)\rangle \otimes |i\rangle \Big),$$

where $T_M(i)$ represents a sub-sequence of the reference T of length M starting at the position i.

The next step in the algorithm evolves the data qubits to their Hamming distances with respect to the search pattern P, which in the case of DNA reconstruction is the short read from the sequencing machines. This operation can be done on the entire superposed state highlighting the parallel transformation power of quantum operators. A set of CNOT gates with the query pattern P as the control on the data qubits results in the data register evolving to the superposition of Hamming distances between each original data and the query pattern. The black-box nature of the oracle function is thus simplified. For a perfect match, the oracle now needs to mark the states with the value of 0, thus making it a fixed function with no dependence on either the reference or the search pattern. Once the state is amplified according to the modified Grover's algorithm (for an unknown number of solutions), the location of the sequence in the database can be determined by making a measurement on the second part of the entangled register, that is, the tag qubits. Note that both the index and data qubits need to be part of the Grover iterations as the two quantum registers are entangled.

For approximate matching, the oracle needs to be modified such that it finds the minimum value of the Hamming distance, instead of an exact 0. We propose an algorithm that merges the improved distributed query approach on the indexed quantum data structure, to retrieve the index of the closest match.

2.3.4. THE PROPOSED QIBAM ALGORITHM

The algorithm presented here inherits some of the features from the approaches highlighted in the algorithms presented in the previous sections. It is a novel quantum pattern matching algorithm specifically designed for the context of genome sequence reconstruction. These requirements for the quantum algorithm are:

- approximate query matching to handle read errors;

- a constant oracle to prevent run-time quantum circuit compilation;
- retrieval of the associated index in the reference along with the corrected query.

The proposed algorithm meets these three requirements. Thus, it can be applied in an implementation for the example illustrated earlier in Figure 2.2. The quantum circuit blocks for the proposed quantum indexed bidirectional associative memory (QiBAM) algorithm is depicted in Figure 2.6.

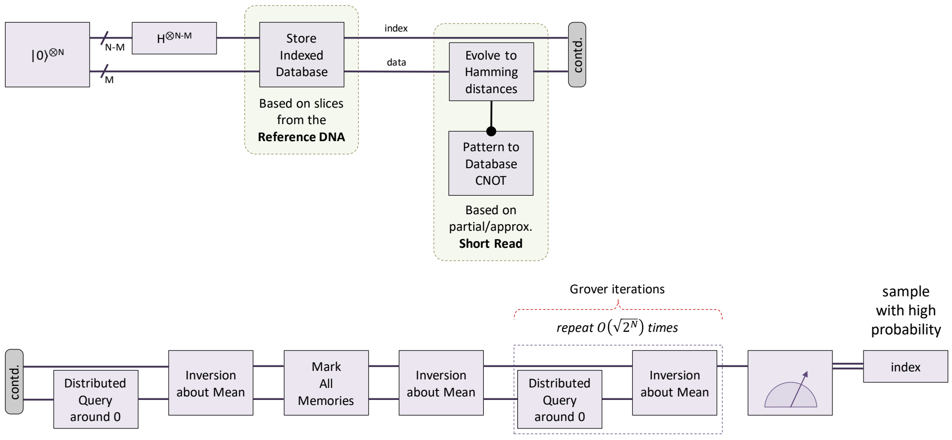


Figure 2.6: Quantum circuit blocks of the proposed QiBAM algorithm.

The initialization of the algorithm follows the design as described in § 2.3.3. The tag qubits encode the pattern index, while the data qubits form the associative memory. Thus, the pattern store step in the associative memory algorithm (refer § 2.3.3) is modeled as a quantum phone directory encoding - which allows the recall of the tagged index corresponding to the query pattern completion/correction. Once the data are encoded, the Hamming distance evolution is carried out. This solves the black-box nature of Grover's marking oracle.

A distributed query is then defined over the associative memory with the query center at zero Hamming distance. This is based on the quantum associative search, now modified with $p = 0$, such that,

$$|b_0^x\rangle = \sqrt{\gamma^{h(0,x)}(1-\gamma)^{d-h(0,x)}}$$

The value of γ is empirically set to 0.25 based on the quantum simulation results. In principle, this free parameter for the application of DNA sequence reconstruction needs to be tuned based on the error rate of the sequencing machine which generates the short read patterns that need to be aligned to the reference DNA.

Thereafter, the minimum Hamming distance (the evolved data string with the largest number of zeros in the basis string in the superposition) is amplified by the process of distributed quantum associative search. Finally, the index qubits are read out to sample with high probability the index where the search pattern best matches the reference.

While the societal relevance of the application presented in this research is enormous, it is important to restate that currently available state of the art quantum processors are not yet capable to implement a proof of concept of this algorithm due to the limitations in the qubit multiplicity, error rates and connectivity. Additionally, since the reference DNA needs to be accessed for each run, as with most quantum algorithms, we assume the availability of a QRAM. Efficient realizations of QRAM is a separate research topic. Additionally, we propose that, if an efficient QRAM implementation is not possible, multiple copies of the QiBAM algorithm can be executed in parallel based on the multiplicity of the qubit, since the result for each search pattern is independent of another. In this context of parallelism, we envisage a multi-core quantum processor where each subset of qubits would be executing the QiBAM on a specific DNA search string similar to how single-instruction-multiple-data (SIMD) is implemented within the cores on a GPUs. Based on an initial version of this research, an extension to Multiple Sequence Alignment has been carried out by [50].

In further sections, a proof-of-concept on classical simulation of quantum computation is shown instead of experimenting with NISQ hardware. It is hard to predict the timeline of quantum processors that will be able to implement this algorithm; however, given the current research thrust and development, a 5–10 years estimate is reasonable [51]. A quantum computer architectural stack aids in developing a quantum algorithm while being agnostic to the underlying hardware, such that, the programs implementing the QiBAM algorithm can readily be ported to any quantum processor once the technological maturity is reached.

Quantum indexed multi-associative memory

If both the quantum registers are accessible for gate operations, the associative memory can be operated (searched) based on either of the registers, thus allowing a bidirectional associative search and retrieval. We can search with the index (in a RAM mode), or by the data (in a CAM mode).

This idea of associative memory can be generalized to multiple quantum registers holding different attributes of the data that needs to be analyzed. This generalization of QiBAM is called quantum indexed multi-associative memory (QiMAM). For example, the quality value of the reads can be stored in register 2, and the chromosome number of the read in register 3, in addition to the index and the pattern. More complex search queries can be formed based on this entangled quantum database, for example, a search for the index of a specific noisy query pattern among high quality reads in a particular chromosome. Such a database is particularly useful for applications like Gene Ontology, Sequence Ontology and Genome Wide Association Studies.

Qubit and gate complexity

Three important parameters are used as metrics for a quantum algorithm. The space and time complexity of classical algorithms, correspond respectively, to the scaling behavior of the number of required qubits and the number of required gates for our quantum computation model. The detailed derivations of the complexity are presented in [29] and here we present the final results. The probability of reading out the desired solution is another important metric though it is dependent on the specific data. Currently, however, it is not possible to use real DNA sequence ensembles in the implementation due to limitations in simulation and available hardware.

The qubit (space) complexity is the aggregate of the qubits used for encoding the data register, tag register and ancilla. We do not consider overheads for error-correction, mapping, routing or other factors besides the algorithm logic. For QiBAM, the qubit complexity is the same as the algorithm in § 2.3.3.

Let the number of qubits required for the data and tag registers be $q_d = \lceil log_2(A) \rceil M$ and $q_t = \lceil log_2(N-M) \rceil$, respectively. The total number of qubits is thus $Q = q_d + q_t + 1$, yielding a typical estimate for the DNA alphabet, the human genome and read length (e.g., from Illumina sequencers) $A = 4$, $N = 3 \times 10^9$ and $M = 50$, as 133 fully-connected logical qubits. While this is beyond the reach of current NISQ era hardware, we note that the number of required qubits to achieve quantum advantage is considerably less than the exemplary Shor's algorithm for factorization for the RSA coding in the cryptography context.

The gate complexity depends on the choice of the universal gate set. Here, the gate set used consists of $\{H, Ry, Rz, C_cX\}$, where $c = 0$ is the X-gate, $c = 1$ is the CNOT gate, $c = 2$ is the Toffoli gate, and so on. Higher-order controls can be decomposed with ancilla qubits [52]. The translation of the gate complexity to the run-time for a specific quantum processor platform would depend on the native gate set available on the hardware. Most modern quantum compilers can convert between universal gate sets in linear compile time and polynomial gate overhead. Thus, the universal gate set considered here is without reference to any specific quantum processor platform. The initialization kernel is first decomposed. First, q_t Hadamard gates are used on the tag qubits to create a superposition of solution states. Then, conditioned on each tag, the corresponding shifted sub-string of the reference is encoded. The binary encoding of the tag requires half the controls as inverted on average, requiring X-gate dressing totaling $q_t 2^{q_t}$. We can use results from the statistical distribution of the nucleotide frequencies to estimate the typical case complexity of the quantum circuits. The Chargaff's rules [53] state that the DNA nucleotides are distributed approximately $1/4$ in each sub-string. This requires $q_d/2$ targets for each tag encoded sub-string. Thus, the total initialization and Hamming evolution require $q_t H + q_t 2^{q_t} C_0 X + q_t q_d / 2 C_{q_t} X$ gates.

The distributed query step depends entirely on the chosen decomposition method for the unitary and the native gate set. The unitary decomposition method using Quantum Shannon Decomposition (QSD) [54] has a complexity of $3(4^{n-1} - 2^{n-1})$, where n is the dimension of the unitary. The mark memory operation would evolve the states in the initial quantum database. This requires the controlled-Z (also called, CPhase) quantum logic gate over the tag and data qubits for each of the 2^{q_t} memories of which $N - M + 1$ are memories from the reference genome. The data qubits, following Chargaff's rule, would have half the bits of 1, thus using a total of M qubits in average for the compute and uncompute. The tag qubits would follow the same behavior as the initialization phase, with average X-dressing of $q_t 2^{q_t} C_0 X$ gates. Thus, the total for the marking memory is $(2^{q_t})\{2H + (M + q_t)C_0 X + C_{q_d+q_t-1} X\}$. Note that this is the oracle complexity of the quantum search, which for all practical implementations, always needs to be considered in addition to the polynomial speedup of query complexity. Finally, the Grover diffusion operator is decomposed to $\{2(q_d + q_t) + 2\}H + 2(q_d + q_t)C_0 X + C_{q_d+q_t-1} X$

gates. The details of our implementation of the unitary decomposition algorithm can be found in [55].

While the exponential reduction in space (qubit) complexity is easy to visualize as proposed in the quantum associative memory architecture, the polynomial speedup is not so pronounced. This is due to the exponential terms in the worst-case analysis for the QSD and binary encoding. Many of these gates can be scheduled in parallel in a quantum processor flattening the complexity. The focus of this research is on the correctness of the algorithm for the presented application, while retaining the oracle query complexity of the Grover search. However, since the oracles are deterministically computable, they can be aggressively optimized by the compiler before run-time.

Tag	Input Database	Input Encoding	XOR with CA = 10	Hamming Distance	Estimate Amplification
0	AA	00	10	1	3
1	AT	03	13	3	1
2	TT	33	23	3	1
3	TG	32	22	2	2
4	GT	23	33	4	0
5	TC	31	21	2	2
6	CT	13	03	2	2
7	TA	30	20	1	3
8	AG	02	12	2	2
9	GG	22	32	3	1
10	GC	21	31	3	1
11	CG	12	02	1	3
12	GA	20	30	2	2
13	AC	01	11	2	2
14	CC	11	01	1	3

Figure 2.7: Quantum database for search pattern *CA* and reference string *AATTGTCTAGGCGACC*.

2.3.5. QIBAM RESULTS ON DNA SEQUENCES

The implementation of this algorithm is carried out in OpenQL [56] and the QX Simulator [57]. In the following example, the results of implementing the QiBAM algorithm on an actual DNA sequence is shown. The details of how to search for a pattern of length 2 over the DNA alphabet (*A, C, G, T*) is demonstrated. A minimum-length super-string that includes all possible length-2 DNA substrings is *AATTGTCTAGGCGACCA*. This minimal-length super-string helps with verifying the correctness of the quantum algorithm exhaustively. To test the distributed query capabilities of the algorithm for mismatches in the reference sequence, the last memory, *CA* is not encoded, making the reference genome as *AATTGTCTAGGCGACC*. This is encoded as the input database shown in Figure 2.7. The radix-4 symbols of the DNA alphabet are encoded in binary as $00, 01, 10, 11$, while the tag is encoded as a 4 bit binary coded decimal value. Thus, the database (of the first two columns in Figure 2.7) is encoded in a quantum superposition as: $|\text{tag\_data}\rangle = a(|0000\_0000\rangle + |0001\_0011\rangle + |0010\_1111\rangle + |0011\_1110\rangle + |0100\_1011\rangle + |0101\_1101\rangle + |0110\_0111\rangle + |0111\_1100\rangle + |1000\_0010\rangle + |1001\_1010\rangle + |1010\_1001\rangle + |1011\_0110\rangle + |1100\_1000\rangle + |1101\_0001\rangle + |1110\_0101\rangle + |1111\_xxxx\rangle)$, where the amplitude $a = \frac{1}{\sqrt{16}}$

Since the tag qubits are in a full superposition, the extra tags (e.g., 1111 in this case) are allocated any arbitrary value (xxxx) and measurements of index beyond the valid range can be ignored.

Now the search query is chosen as *CA*. The search pattern conditionally toggles the database to evolve it to the Hamming distance. Since *CA* is not present in memory, we expect the nearest patterns (approximate matches) to have a higher probability of detection, which are *{AA, TA, CG, CC}* (with a Hamming distance of 1 in the encoding). This results in the superposition: $|\text{tag\_dist}_{\text{Ham}}\rangle = a(|0000\_0100\rangle + |0001\_0111\rangle + |0010\_1011\rangle + |0011\_1010\rangle + |0100\_1111\rangle + |0101\_1001\rangle + |0110\_0011\rangle + |0111\_1000\rangle + |1000\_0110\rangle + |1001\_1110\rangle + |1010\_1101\rangle + |1011\_0010\rangle + |1100\_1100\rangle + |1101\_0101\rangle + |1110\_0001\rangle + |1111\_xxxx\rangle)$. It can be verified for this case that, at indices $0000, 0111, 1011, 1110$, the dist_{Ham} quantum register has the minimum number of 1 s and thus will be amplified by a distributed query around 0000 followed by Grover diffusion for the required number of iteration. The estimated trend for a higher solution probability should be in line with decreasing Hamming distance, as plotted in Figure 2.8, with the tag on the X-axis and the Estimate Amplification on the Y-axis.

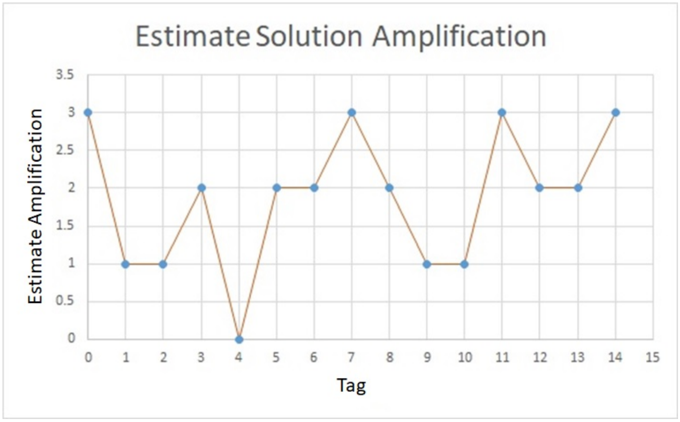


Figure 2.8: Solution probability as a numerical estimate of expected results of a sample run.

The OpenQL algorithm is executed with the Qxelarator library, returning the internal state vector. The reference sequence and the search query is hardcoded in the Python program for this test but can be streamlined to be directly read from an industry-standard file like the FASTQ format from commercial DNA sequencers. The result from the run is plotted in Figure 2.9. The left vertical axis shows the staircase state curve for the tag qubits, while the right vertical axis shows the measurement probability of each individual state. There are four tag qubits and four data qubits (2 radix-4 numbers for a DNA search pattern of length 2). Thus, the total state space is $2^8 = 256$. The double-precision floating point naive state vector simulation of this algorithm requires 32 Kb to store the state space, while each of the ≈230 gates requires a matrix of 8Mb. The states with prominent probabilities are the memory states. The envelope of these states (ignoring the spurious memories) gives the same trend as our estimate in Figure 2.8, verifying the correctness of our implementation.

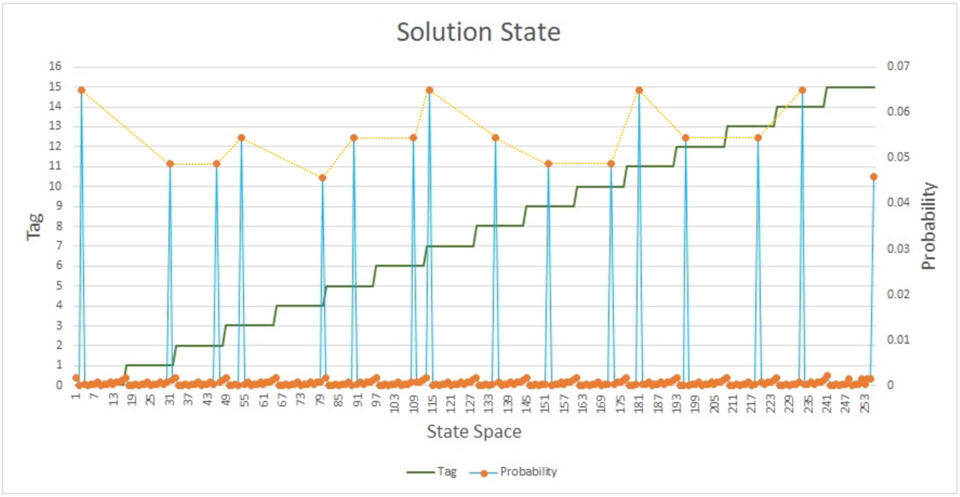


Figure 2.9: Results of a sample execution of QiBAM on QX Simulator. The trend line (in orange) of the probability distribution (after filtering the spurious states), matches the results derived analytically in Figure 2.8.

2.4. QUANTUM-ACCELERATED READ ASSEMBLY

In this research, we explore the alternative reference-free DNA sequence reconstruction implementation via de novo assembly, as an implementation on a quantum computing platform. De novo assembly using the overlap-layout-consensus method has many advantages over other simpler methods, but suffers from large computational complexity, which motivates targeting a quantum accelerator. Both gate-based quantum system as well as quantum annealers are targeted. Each step of the formulation is explained with simple examples [58] to target both the genomics research community and quantum application developers. The implementation is evaluated on the D-Wave simulator, the D-Wave annealer in the cloud and the QX Simulator. The current limitations in solving real problem sizes to achieve a quantum advantage are discussed thereafter. It is the first time this important computational problem of de novo assembly in bioinformatics is targeted on a quantum accelerator with the full description of the pipeline.

Recently, [59] also discussed de novo sequencing on quantum annealing and quantum-inspired annealing, citing a preprint of our research as presented in here. We appreciate this research done independently to ours echoing a similar motivation to explore the applicability of quantum computing to DNA sequence reconstruction and reaching similar results. In contrast to their work, we additionally target the gate based quantum computing model, which is considered to be the future standard of quantum acceleration. Furthermore, our focus is from the perspective of a quantum application developer, with details from genomics, quantum and computer science formulated in a self-contained matter. In addition, an example is implemented via the various steps of the algorithm and the execution results are demonstrated as a proof of concept to evaluate the quality, scalability and limitations.

2.4.1. DE NOVO REFERENCE-FREE ASSEMBLY

During sequencing, multiple copies of the DNA are made before fragmenting it. Thus, a portion of the data is preserved in multiple copies which are chopped off at different places resulting in data overlaps which facilitate stitching. This method is called de novo assembly, as no other data than the sequenced read is used for reconstruction. It is computationally expensive and done normally for the first time a new species is sequenced. Given enough computing power, de novo sequencing is highly desirable as it is free of reference bias. Since quantum computing presents itself as an ultimate computing machine, it is worth exploring the problem of de novo sequencing.

There are different methods [60] for de novo assembly used by the available tools: Overlap-Layout-Consensus (OLC) methods, de Bruijn graph (DBG) methods, string graphs, greedy and hybrid algorithm, etc. Real-world WGS data induces various problems in all these methods. Examples are spurs (short, dead-end divergences from the main path), bubbles (paths that diverge then converge), frayed rope pattern (paths that converge then diverge) and cycles (paths that converge on themselves) [61]. Common causes of these complexities are attributed to repeats in the target and sequencing error in the reads. Most optimal graph reductions belong to the NP-hard class of problems, thus assemblers (like Euler, Velvet, ABySS, AllPaths, SOAPdenovo) rely on heuristics to remove redundancy, repair errors or otherwise simplify the graph. The choice of algorithms is based on the quality, quantity and structure of the genome data. Current short-read sequencing technologies produce very large numbers of reads favoring DBG methods. However, single molecule sequencing from third generation sequencing machines produces high-quality long reads, which could favor OLC methods.

In DBG, the nodes represent all possible fixed-length strings of length K (K-mer graph). The edges represent fixed-length suffix-to-prefix perfect overlaps between subsequences that were consecutive in the larger sequence. In WGS assembly, the K-mer graph represents the input reads. Each read induces a path and those with perfect overlaps induce a common path as an advantage, however, compared to overlap graphs, K-mer graphs are more sensitive to repeats and sequencing errors as K is much less than read size. In an ideal construction, the Eulerian path corresponds to the original sequence, though graphs built from real sequencing data are more complicated.

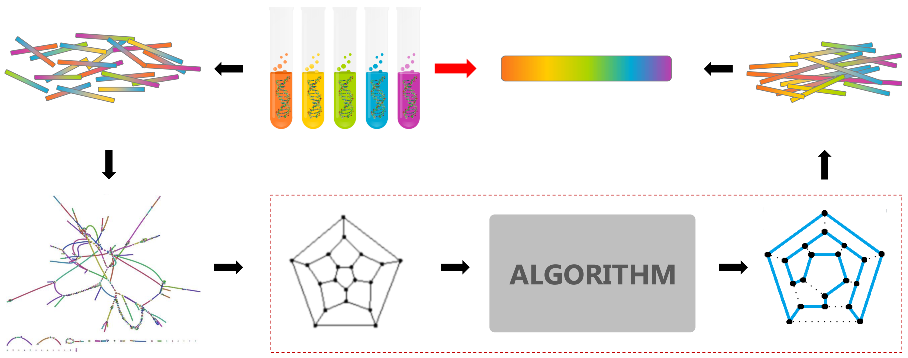


Figure 2.10: Overlap-Layout-Consensus genome assembly algorithm.

In the OLC method [62], as shown in Figure 2.10, an overlap graph represents the sequencing reads as nodes and their overlaps (pre-computed by pair-wise sequence alignments) as edges. Paths through the graph are the potential assembled DNA pieces and can be converted to sequence. Formally, this represents a Hamiltonian cycle, a path that travels to every node of the graph exactly once and ends at the starting node, including each read once in the assembly. There is no known efficient algorithm for finding a Hamiltonian cycle as it is in the NP-complete class. Though it was feasible for microbial genome (in 1995) and the human genome (in 2001), NGS projects have abandoned it due to the high computational burden to be commercially viable. This is the target for quantum acceleration in this research.

2.4.2. QUANTUM-ACCELERATED OPTIMIZATION

The OLC method of de novo assembly requires solving the minimum Hamiltonian cycle problem. This is also famously known as the Traveling Salesman Problem (TSP), which belongs to the NP-hard class of computational complexity. Thus, we need to formulate it as an approximate optimization problem as even quantum computers cannot solve NP-hard problems in polynomial time.

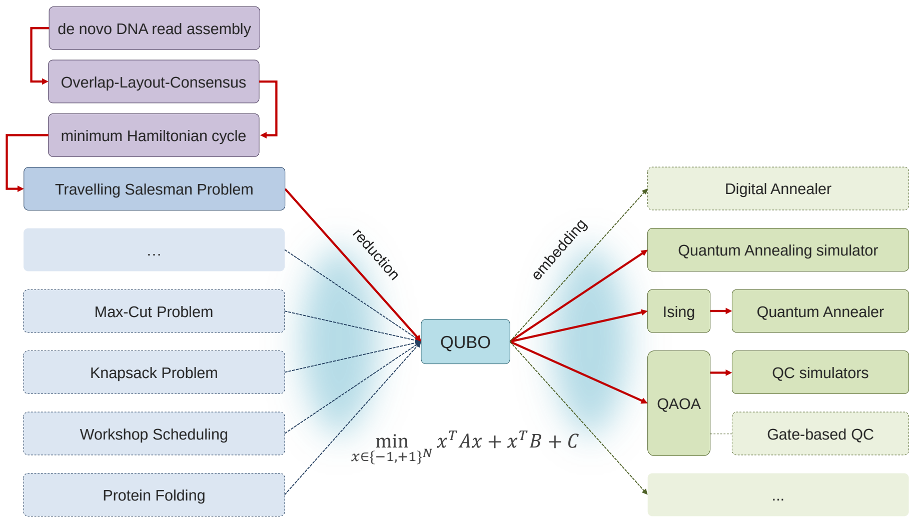


Figure 2.11: Workflow (red arrows) of quantum accelerated read assembly.

Mapping an NP-hard problem to quantum involves 2 steps as shown in Figure 2.11. The first step is to reduce the given application to a Quadratic Unconstrained Binary Optimization (QUBO). The second step is to embed the QUBO to the connectivity structure of the hardware. Worst-case run-time for NP-Hard problems are exponential even on a quantum computer. The aim of the physical implementations (like QAOA and quantum annealers from D-Wave systems) is to try to find a good approximation to the solution in polynomial running time. The technique to formulate a problem and program the corresponding quantum computing model of gate-based and annealers are considerably different. This is presented in the next sections.

QUADRATIC UNCONSTRAINED BINARY OPTIMIZATION

A binary quadratic model (BQM) comprises a collection of binary-valued variables that can be assigned two chosen values (based on the model) with associated linear and quadratic biases. Two isomorphic BQM are:

- QUBO models: $x_i \in \{0,1\}$ Boolean values

$$H(x) = \sum_i Q_{ii} x_i + \sum_{i,j} Q_{ij} x_i x_j + k$$

- Ising models: $\sigma_i \in \{-1,+1\}$ spin states

$$H(\sigma) = -\mu \sum_i h_i \sigma_i - \sum_{i,j} J_{ij} \sigma_i \sigma_j$$

The choice of model depends on the problem. Using QUBO, it might be easier to write numbers in standard binary notation (e.g. $101_2 = 5_{10}$) in the optimization problem; or it might be required to destructively interfere two variable (e.g. two battling Pokemons) using the Ising model. Quantum processors like annealers use the Ising model, thus QUBO equations need to be converted into Ising under the hood. QUBOs can always be fully expressed in both expanded and matrix forms, while Ising can be fully expressed in the expanded form, but not completely in the matrix form. However, the being isomorphic, they are equivalent in principle, and the choice depends on the underlying hardware implementation or ease of expressing the logic.

QUBO model [63, 64] unifies a rich variety of combinatorial optimization problems as an alternative to traditional modeling and solution methodologies. These problems are concerned with making wise choices in settings where a large number of yes/no decisions must be made and each set of decisions yields a corresponding objective function value - like a cost or profit value. The QUBO model is expressed by the optimization problem:

$$\texttt{minimize} \;\; y = x^t Q x = H(x)$$

where x is a vector of binary decision variables and Q is a (symmetric or in upper triangular) square matrix of constants.

Different types of constraining relationships arising in practice can be embodied within the unconstrained QUBO formulation using penalty functions. The penalties introduced are chosen so that the influence of the original constraints on the solution process can alternatively be achieved by the natural functioning of the optimizer as it looks for solutions that avoid incurring the penalties. Penalties are not unique, meaning that many different values can be successfully employed. For a particular problem, a workable value is typically set, based on domain knowledge and on what needs to be accomplished. If a constraint must be satisfied, i.e., a hard constraint, then the penalty must be large enough to preclude a violation. More moderate penalties are set for soft constraints, meaning that it is desirable to satisfy them but slight violations can be tolerated. Casting the QUBO model as a minimization problem permits a maximization problem to be solved by minimizing the negative of its objective function.

QUBO models belong to the NP-hard class of problems. Thus exact solvers (e.g. CPLEX, Gurobi) work practically only for very small problem instances (around 100 variables) using mostly branch-and-bound or problem-specific techniques. However, impressive successes are being achieved by using meta-heuristic methods that are designed to find high quality but not necessarily optimal solutions in a modest amount of computer time. Among the best meta-heuristic methods for QUBO are those based on tabu search, path relinking, simulated annealing, genetic/memetic strategies, and their ensembles. Recently, with the availability of small-scale quantum processors, there is a huge research thrust in achieving quantum advantage for QUBO models.

TRAVELING SALESMAN PROBLEM

A Hamiltonian path is a graph path between two vertices of a graph that visits each vertex exactly once. If a Hamiltonian path exists whose endpoints are adjacent, then the resulting graph cycle is called a Hamiltonian cycle. It is a path that starts from one node and ends at the same node covering all the nodes of that graph.

Given a directed complete graph $G = (V, E)$ with weights w_{ij} on the directed edge $i \rightarrow j$, the directed traveling salesman problem (TSP) aims to find a directed Hamiltonian cycle of minimum weight, i.e., a cycle that visits all nodes (cities) of the graph and such that the sum of the edge weights (travel cost) is minimum. Intuitively, given the ordered pair-wise distance between cities, the TSP involves finding the shortest route that visits every city once. The order of visiting the cities are not constrained.

TSP falls under the NP-hard class (thus outside BQP), so the time to find the exact solution scales exponentially also on a quantum computer with the problem size. Often a good sub-optimal solution is admissible, thus heuristic algorithms of much lesser complexity can be employed. TSP solvers are used in many industrial applications in the domains of planning, scheduling, logistics, packing, DNA sequencing, network protocols, telescope control, VLSI testing, and many more.

The first step to specifying a TSP is to create a (weighted) graph specifying the edges in the format (vertex-from; vertex-to; weight). Next, the TSP graph is transformed into a QUBO graph. QUBO variables are labeled (n, t) where n is a node (read) in the TSP graph and t is the time index of visiting it in order. E.g., if $(a, 0) = 1$ in the solution state, that means the node a is visited first. Since the total number of visits (time IDs) equals the total number of nodes (read IDs); the total possible combinations of (n, t) is $|G|^2$. $|G|$ is the number of nodes in the original TSP graph.

The QUBO graph will have $2*|G|^2*(|G|-1)$ interactions (or edges). The interactions denote pairs of 2 nodes that can/cannot coexist. The weight of the interaction shows the reward/penalty of coexisting. A higher positive value denotes more penalty. There are 3 types of penalty, for multi-location (being at 2 places at the same time), repetition (being at a city twice) and path cost for the tour.

HAMILTONIAN FORMULATION

In physical systems (classical or quantum), a Hamiltonian describes the energy of an object. More specifically, it describes the time-evolution of a system expressed by the Schrödinger equation:

$$i\hbar \frac{d}{dt}|\psi(t)\rangle = H|\psi(t)\rangle$$

The unitary operator underlying the Hamiltonian is obtained by solving the equation for some time duration: $U = \exp(-iHt/\hbar)$. The time-independent formulation of the equation reflects the total energy of the system $E = H|\psi\rangle$.

The adiabatic theorem dictates that if the change in the time-dependent Hamiltonian occurs slowly, the resulting dynamics remain simple, i.e. starting close to an eigenstate, the system remains close to an eigenstate. For a quantum mechanical system, some initial Hamiltonian H_i is slowly changed to some other final Hamiltonian H_f. This implies that, if the system is started in the ground state (lowest eigenstate) of the initial Hamiltonian, the system will evolve to the ground state of the final configuration. The computational advantage comes from the choice of an easy-to-prepare quantum system like:

$$H_i = -\sum_i \sigma_i^X$$

(the ground state is the equal superposition state) and evolve the Hamiltonian to a system such that the ground state encodes the solution of the optimization problem we are interested in.

The change needs to be carried out by a defined schedule, for example, linear in the time scale $t \in [0,1]$ defined as:

$$H(t) = (1-t)H_i + tH_f$$

The energy difference between the ground state and the first excited state is called the gap, $\Delta(t)$. If $H(t)$ has a finite gap for each t during the transition the system can be evolved adiabatically with the evolution speed proportional to $1/\min(\Delta(t))^2$. The gap, however, is highly problem-dependent, tending to have an exponentially small gap for hard problems (like those in the NP-hard class), making the time exponentially long. Thus it is unlikely that an exact solution for these problems can be found in polynomial time.

In adiabatic quantum computations, universal calculations are performed by mapping the problem to a final Hamiltonian defined as:

$$H_f = -\sum_{<i,j>} J_{ij}\sigma_i^Z\sigma_j^Z - \sum_i h_i\sigma_i^Z - \sum_{<i,j>} g_{ij}\sigma_i^X\sigma_j^X$$

Thus the system Hamiltonian $H(t)$ becomes:

$$H(t) = (1-t)\Big[-\sum_i \sigma_i^X\Big] + t\Big[-\sum_{<i,j>} J_{ij}\sigma_i^Z\sigma_j^Z - \sum_i h_i\sigma_i^Z - \sum_{<i,j>} g_{ij}\sigma_i^X\sigma_j^X\Big]$$

The values of biases and couplings are set by the user/programmer for a quantum annealer.

The major drawback to implementing an adiabatic quantum computing directly is calculating the speed limit, which is harder than solving the original problem of finding the ground state of a Hamiltonian. Quantum annealing [65] drops the strict requirements of respecting speed limits in favor of repeating the transition multiple times. Sampling from the solutions is likely to find the lowest energy state of the final Hamiltonian (though there is no theoretical guarantee). Going from ‘nearly correct’ to ‘correct’ is still NP in general if the original problem is in the NP class of complexity (the parameters for local optima aren’t necessarily going to be distributed anywhere near the global optima). However, annealing can be useful if a sub-optimal solution is acceptable for the application.

VARIATIONAL HYBRID APPROACH

Coherent quantum protocols have promising exponential speedups but assume a fault-tolerant quantum computing platform, with high quality of qubits, a large number of gates and circuit width. These popular algorithms like Shor's factorization, HHL for matrix inversion, though primitives for many quantum algorithms are not of immediate practical relevance. Wrapping these protocols in state preparation (from classical data to quantum) and state tomography (from quantum probability amplitudes of final state to classical statistics) can overrule the entire speedup achieved by the protocol itself.

Currently, we are in the NISQ era. These near-term quantum computers are more suited for hybrid quantum-classical (HQC) algorithms [66]. These are heuristic protocols based on the variational principle. In an HQC algorithm, the power of the quantum computer is used for preparing a quantum state. The complexity of the algorithm is traded off for multiple measurements over multiple cycles. The operations that require lots of gates on a quantum computer are offloaded to the classical computer (e.g. optimization, addition, division), which controls the quantum computer like an accelerator or a co-processor, as shown in Figure 2.12. A quantum circuit is defined as having a certain format A (or ansatz/stencil) with parameters. There are m parameters forming a parameter vector Λ_m. These can be initialized randomly or with a classical guess. For the first cycle, the quantum computer takes the initial guess and evolves it using the circuit $A(\Lambda_m^0)$. The Hamiltonian (energy) is measured out and sent to the classical computer. The variational principle updates the parameters in such a way that the energy of the Hamiltonian is lowered in each successive iteration. The optimization using $\Lambda_m^1, \Lambda_m^2, \Lambda_m^3, \ldots$ continues until the acceptable threshold is satisfied, very similar to training in neural networks.

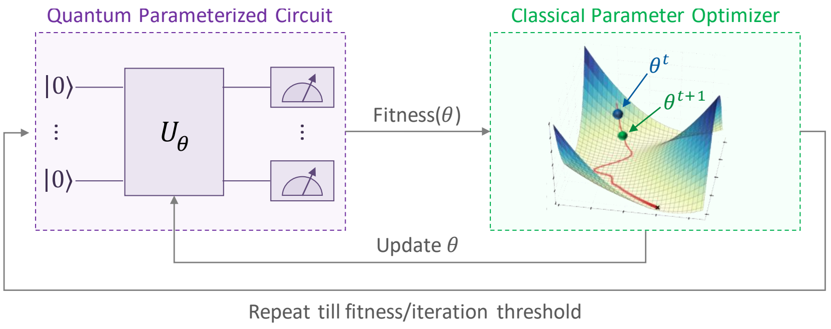

Quantum Parameterized Circuit
Classical Parameter Optimizer
$|0\rangle$
$|0\rangle$
U_θ
Fitness(θ)
θ^t
θ^{t+1}
Update θ
Repeat till fitness/iteration threshold

Figure 2.12: Variational hybrid quantum-classical approach for optimization.

The variational principle forms the core theoretical basis behind the working of near-term quantum heuristic algorithms. It states that, for a trial wave-function (defining the family of quantum states reachable by varying the m parameters of A),

$$E_{\Psi_T} = \frac{\langle \Psi_T(\Lambda) | \hat{H}_{QC} | \Psi_T(\Lambda) \rangle}{\langle \Psi_T(\Lambda) | \Psi_T(\Lambda) \rangle} \geq E_0$$

The normalization term $\langle \Psi_T(\Lambda) | \Psi_T(\Lambda) \rangle = 1$ as we assume no leakage errors from the computational basis. Thus, it is possible to reach the ground-state energy by finding the right parameters. The more free the state is to represent quantum states (determined by the choice of A), the better it will be able to lower the energy.

Since adiabatic and gate systems offer effectively the same potential for achieving the gains inherent in quantum computing processes, analogous advances associated with QUBO models may ultimately be realized through quantum circuit systems as well.

An example of HQC algorithms is the Quantum Approximate Optimization Algorithm (QAOA) [67]. It is a hybrid variational algorithm that produces approximate solutions for combinatorial optimization problems. In theory, QAOA methods can be applied to more types of combinatorial optimization problems than embraced by the QUBO model [64]. The parameters of the QAOA framework must be modified to produce different algorithms to appropriately handle different problem types. QAOA is a polynomial-time HQC algorithm which can be seen as the Trotterization of an infinite time adiabatic algorithm. Since the AQC always gives the optimal solution for Hamiltonians with a non-zero gap, QAOA for infinite cycles also converges to the global optima.

The generalization of QAOA called the Quantum Alternating Operator Ansatz [68], consists of 2 Hamiltonians: a cost/problem Hamiltonian H_C (similar to the transverse field in AQC) and a driver/mixing Hamiltonian H_M (similar to the longitudinal field in AQC). This is repeated over p cycles with each Hamiltonian parameterized by the γ and β real values (rotation angles similar to the adiabatic evolution time). After this unitary evolution, the state is measured for the expectation value with respect to the ground state of the cost Hamiltonian. The initial state $|\psi_0\rangle$ depends on the problem (typically either the all-zero or the equal superposition state).

$$U(\theta)\,|\psi_0\rangle = H_M(\beta_p)H_C(\gamma_p)\dots H_M(\beta_2)H_C(\gamma_2)H_M(\beta_1)H_C(\gamma_1)\,|\psi_0\rangle$$

For an optimization instance, the user specifies the driver Hamiltonian ansatz, cost Hamiltonian ansatz, and the approximation order (cycles) of the algorithm. If the number of cycles in QAOA increases, theoretically, the sub-optimal solutions obtained can only get better, as the sub-optimal solutions defined by fewer cycles (with fewer free parameters) are always contained in more cycles (if the new rotation parameters are set to zero). However, practically, having more cycles causes difficulty for the classical optimizer to deal with more free parameters and can affect the convergence.

Since HQC trades off the decoherence issue of a long quantum circuit in the NISQ era with multiple low-depth, the number of repetitions required is high. To estimate the expectation of the prepared state in each optimization step, it needs to be (pre-rotated in the basis and) measured with respect to each Pauli term in the problem Hamiltonian and aggregated. Each Pauli term measurement in turn requires state tomographic trials.

Moreover, the optimizer might get stuck in local optima or barren plateaus [69] in the parameter landscape, requiring a few reruns to build confidence in the obtained optima. The HQC algorithms depend a lot on the choice of the classical optimizer as well. Here we experiment with the basic Nelder-Mead gradient-free optimizer, but many gradient-based and gradient-free choices exist (for example in libraries like SciPy in Python and TOMLAB in MATLAB) which needs to be chosen based on empirical testing of a particular formulation of the specific problem.

The pseudo-code for our OpenQL implementation of the QAOA algorithm is shown in Listing 2.1.

```
• Invoke QAOA:
  ◇ Form parameterized cQasm [Input: reference circuit, ansatz, steps, coefficients, angle ids]
  ◇ Form parameters list [Input: gammas, betas]
  ◇ Set runtime deparameterized cQasm filename
  ◇ For each iteration:
    ⋆ For each Pauli product term in cost Hamiltonian (wsopp):
      ∘ Deparameterize cQasm [Input: parameterized QASM, parameter list]
      ∘ Add measurement basis rotation based on Pauli product term
      ∘ Invoke Qxelerator to aggregate measurement over shots
    ⋆ Calculate total Expectation value of trail state in cost Hamiltonian basis
    ⋆ Return value to optimizer. Save intermediate results via callback.
    ⋆ Classical optimizer updates parameter list
  ◇ Display cost function convergence, final parameters and optimized cost
```

Listing 2.1: Pseudo-code for OpenQL QAOA.

2.4.3. IMPLEMENTING DE NOVO ASSEMBLY

In this section, first, we present a mapping from DNA to TSP, as common in the OLC method. Then the TSP is converted to a QUBO. A fully worked out example is presented as a proof of concept for quantum accelerated sequence reconstruction. The example is chosen based on the limit of currently available quantum computing hardware and classical simulators, however, the formulation and implementation is generic and scalable to industrial level pipelines.

DNA READS TO TSP FORMULATION

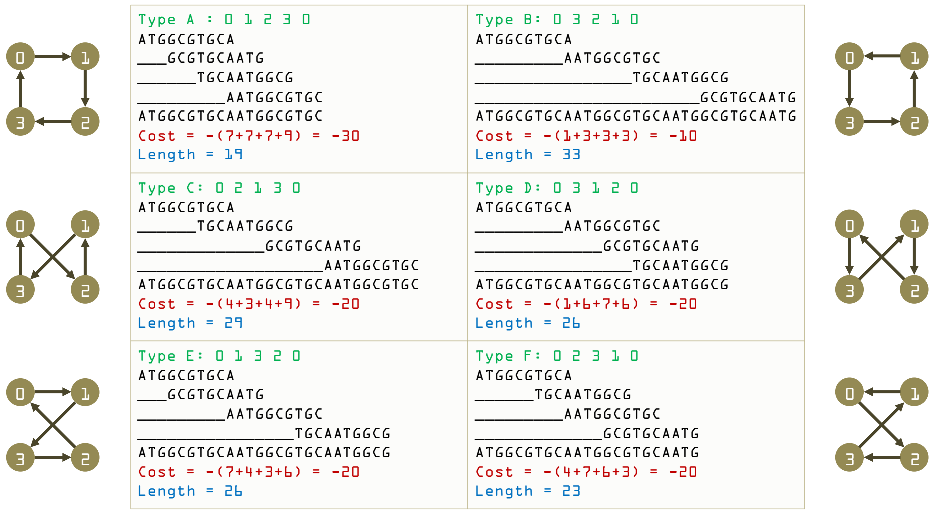
Suppose the sequencer produces reads of size 10 nucleotide bases and after removing duplicates, the reads obtained are:
read 0 : ATGGCGTGCA
read 1 : GCGTGCAATG
read 2 : TGCAATGGCG
read 3 : AATGGCGTGC

The pairwise overlap is calculated for each ordered pair based on how many prefix characters of the second string match exactly with the suffix of the first string. The edge weight is set to the negation of the overlap, as the constraints need to be formulated such that the path that *minimizes the overlap* is found.

The edge weights are *{(0,1):-7 , (1,2):-7 , (2,3):-7 , (3,0):-9 , (1,0):-3 , (2,1):-3 , (3,2):-3 , (0,3):-1 , (0,2):-4 , (1,3):-4 , (2,0):-6 , (3,1):-6}*

The overlap depends on the ordering of the read pairs and thus this formulation is a directed graph. There are 6 possible unique tours (choosing a different starting city in the tour is equivalent in cost). Note that the reads are spliced from an original circular DNA, so the final stitched DNA solutions for all these tours are repeats of *read 0* of variable length. Such cases occur in practice when arranging the reads in different ways gives different repeat lengths. The tours are emulated in Figure 2.13. The TSP solution is expected to find the lowest cost (and shortest assembly) tour, i.e. Type-A. There are 4 acceptable solutions (based on starting node) of Type-A:

- $(0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 0)$
- $(1 \rightarrow 2 \rightarrow 3 \rightarrow 0 \rightarrow 1)$



`Type A : 0 1 2 3 0`
`ATGGCGTGCA`
`___GCGTGCAATG`
`______TGCAATGGCG`
`_________AATGGCGTGC`
`ATGGCGTGCAATGGCGTGC`
`Cost = -(7+7+7+9) = -30`
`Length = 19`
	`Type B: 0 3 2 1 0`
	`ATGGCGTGCA`
	`_________AATGGCGTGC`
	`________________TGCAATGGCG`
	`_______________________GCGTGCAATG`
	`ATGGCGTGCAATGGCGTGCAATGGCGTGCAATG`
	`Cost = -(1+3+3+3) = -10`
	`Length = 33`

`Type C: 0 2 1 3 0`
`ATGGCGTGCA`
`______TGCAATGGCG`
`_____________GCGTGCAATG`
`___________________AATGGCGTGC`
`ATGGCGTGCAATGGCGTGCAATGGCGTGC`
`Cost = -(4+3+4+9) = -20`
`Length = 29`
	`Type D: 0 3 1 2 0`
	`ATGGCGTGCA`
	`_________AATGGCGTGC`
	`_____________GCGTGCAATG`
	`________________TGCAATGGCG`
	`ATGGCGTGCAATGGCGTGCAATGGCG`
	`Cost = -(1+6+7+6) = -20`
	`Length = 26`

`Type E: 0 1 3 2 0`
`ATGGCGTGCA`
`___GCGTGCAATG`
`_________AATGGCGTGC`
`________________TGCAATGGCG`
`ATGGCGTGCAATGGCGTGCAATGGCG`
`Cost = -(7+4+3+6) = -20`
`Length = 26`
	`Type F: 0 2 3 1 0`
	`ATGGCGTGCA`
	`______TGCAATGGCG`
	`_________AATGGCGTGC`
	`_____________GCGTGCAATG`
	`ATGGCGTGCAATGGCGTGCAATG`
	`Cost = -(4+7+6+3) = -20`
	`Length = 23`

Figure 2.13: All possible TSP tours for given example.

- $(2 \rightarrow 3 \rightarrow 0 \rightarrow 1 \rightarrow 2)$
- $(3 \rightarrow 0 \rightarrow 1 \rightarrow 2 \rightarrow 3)$

This process is automated in the classical pre-processing `align` and `reads_to_tspAdjM` functions in our implementation. It can be invoked as

```
reads = ['ATGGCGTGCA', 'GCGTGCAATG', 'TGCAATGGCG', 'AATGGCGTGC']
tspAdjM = reads_to_tspAdjM(reads)
```

TSP TO QUBO MODEL

Next, the directed TSP is encoded as a QUBO model. Let $n = |V| = 4$ be the number of nodes. This formulation [70] requires $n^2 = 16$ binary variables as qubits, so it scales quadratically rather than linearly in the problem size. For $i, p \in \{0\dots(n-1)\}$, let $x_{i,p}$ be True if node i appears in position p in the cycle, False otherwise.

To derive a Hamiltonian for this problem, we penalize the violation of the constraints in the objective function inserting terms of the form $\alpha(\sum_{p=0}^{n-1} x_{i,p} - 1)^2$, where the penalty term is sufficiently large, e.g., $\alpha = n * max_{(i,j)\in E} w_{ij}$. The TSP can be formulated as:

$$\forall i, p \in \{0\dots(n-1)\}, x_{i,p} \in \{0,1\}$$

The interactions are shown in Figure 2.14 and are categorized as:

1. Every node must be assigned. Thus self-interactions have large negative weight (favorable bias). Since there are no preferred order of the route, for each time slot, the value is the same (top-left blue interactions).
2. Same node assigned to two different time slots incurs a penalty (top-middle violet

interactions). Thus, for each node, there should be only one assigned time slot:

$$\forall i \in \{0\dots(n-1)\}, \sum_{p=0}^{n-1} x_{i,p} = 1$$

3. Same time slot assigned to two different nodes incurs a penalty (top-right violet interactions). Thus, for each time slot, there should be only one assigned node:

$$\forall p \in \{0\dots(n-1)\}, \sum_{i=0}^{n-1} x_{i,p} = 1$$

4. The additional cost of including an edge in the route to two consecutive time slots is the weight of the edge in the TSP. These 6 graphs show the 4 possible routes for each of the 6 types (type A in middle-left green, others in red). Each edge (i,j) is taken and all possible configurations of assigning them next to each other are tried (the addition being modulo n), with the edge weight being the cost of choosing from those configurations. Thus, given the above constraints:

$$\texttt{minimize:} \quad \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} w_{ij} \sum_{p=0}^{n-1} x_{i,p}x_{j,p+1}$$

These arrows can be made into an adjacency matrix for the graph, resulting in the Q-matrix as shown in Figure 2.15 (with the colors of the cells representing the terms from the corresponding colored arrows). The addition of these 6 matrices gives the Q matrix for the QUBO. Note that, if we have only the lower 6 matrices (coupling), the all 1's assignment is the most favorable and gives the minimum solution to the QUBO equation $y = x^T Qx$. Thus, we need to add the reward for assigning a node $\{a\}$ and penalties $\{b,c\}$ for assigning the same node to multiple different time slots, or same time slots to multiple nodes, respectively. Since these are bi-directional arrows, these can be symmetric.

For our experiment, we empirically found that setting a penalty value of $b = c \geq 13$ is sufficient, and the reward $a = 0$ still finds the 4 favorable minima of Type A. It is easy to verify that a minimum value of $y = x^T Qx$ is obtained for:

- $x^T = [1000010000100001]$
- $x^T = [0100001000011000]$
- $x^T = [0010000110000100]$
- $x^T = [0001100001000010]$

for the binary encoding

$$x^T = [n_0t_0|n_0t_1|n_0t_2|n_0t_3|n_1t_0|n_1t_1|n_1t_2|n_1t_3|n_2t_0|n_2t_1|n_2t_2|n_2t_3|n_3t_0|n_3t_1|n_3t_2|n_3t_3]$$

The process of generating the Q matrix is automated in the function `tspAdjM_to_quboAdjM` in `../denovo_009.py`. It can be invoked with the adjacency matrix from the `reads_to_tspAdjM` function as

```
# Parameters: adj matrix, self-bias, multi-location, repetation
quboAdjM = tspAdjM_to_quboAdjM(tspAdjM, 0, 13, 13)
```

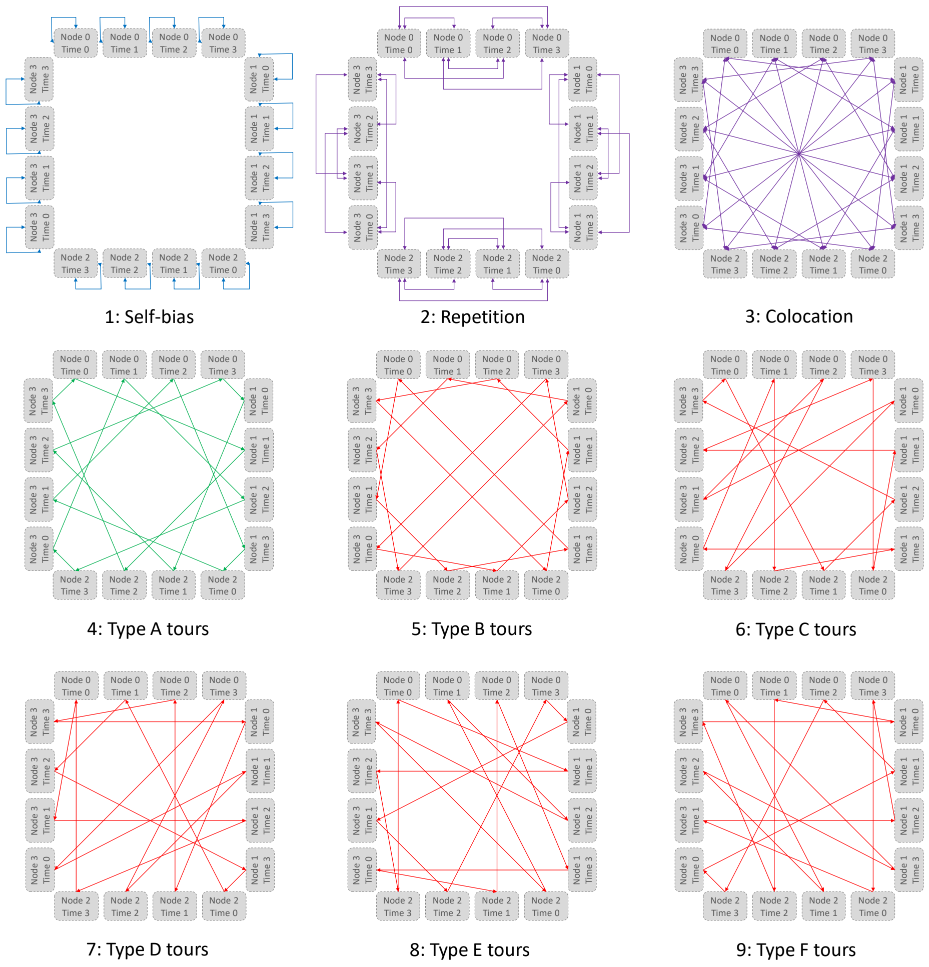


Figure 2.14: QUBO interactions.

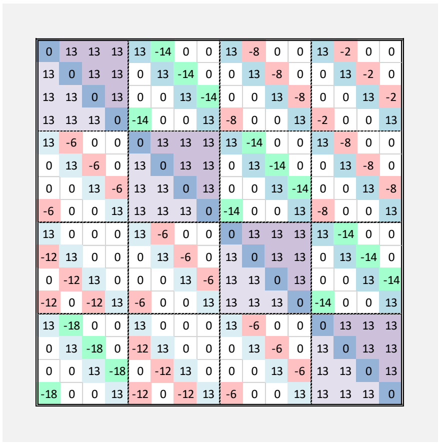
2.4.4. SOLVING ON A QUANTUM SYSTEM

In this section, we show how to solve the QUBO on both a quantum annealer (D-Wave Ocean tools) and a gate-based optimizer (QX/OpenQL) using QAOA.

QUBO USING QUANTUM ANNEALING

The QUBO is mapped to a quantum annealer using the biases and couplings in the Ising model. A bias value is defined for each qubit and a coupling for each pair of qubits. In the graph view, each node (bias) and each edge (coupling) can have a real-number value (weight). Since this example requires 16 QUBO variables (qubits), the exact solver is used to better understand the output.

For the de novo example, the Q matrix from (with $a = 0, b = c = 13$) is shown in Figure 2.15.



0	13	13	13	13	-14	0	0	13	-8	0	0	13	-2	0	0
13	0	13	13	0	13	-14	0	0	13	-8	0	0	13	-2	0
13	13	0	13	0	0	13	-14	0	0	13	-8	0	0	13	-2
13	13	13	0	-14	0	0	13	-8	0	0	13	-2	0	0	13
13	-6	0	0	0	13	13	13	13	-14	0	0	13	-8	0	0
0	13	-6	0	13	0	13	13	0	13	-14	0	0	13	-8	0
0	0	13	-6	13	13	0	13	0	0	13	-14	0	0	13	-8
-6	0	0	13	13	13	13	0	-14	0	0	13	-8	0	0	13
13	0	0	0	13	-6	0	0	0	13	13	13	13	-14	0	0
-12	13	0	0	0	13	-6	0	13	0	13	13	0	13	-14	0
0	-12	13	0	0	0	13	-6	13	13	0	13	0	0	13	-14
-12	0	-12	13	-6	0	0	13	13	13	13	0	-14	0	0	13
13	-18	0	0	13	0	0	0	13	-6	0	0	0	13	13	13
0	13	-18	0	-12	13	0	0	0	13	-6	0	13	0	13	13
0	0	13	-18	0	-12	13	0	0	0	13	-6	13	13	0	13
-18	0	0	13	-12	0	-12	13	-6	0	0	13	13	13	13	0

Figure 2.15: Q-matrix for de novo example.

The Q matrix is converted to a dictionary of node names and reward/penalty for biases and couplings. Then the QUBO solver is used to solve the Q matrix using the assignment of $\{0,1\}$ (instead of the Ising $\{-1,+1\}$). This is coded in the `quboAdjM_to_quboDict` and `solve_qubo_exact` functions in `../denovo_009.py`. It can be invoked with the qubo adjacency matrix from the `tspAdjM_to_quboAdjM` function as

```
Q = quboAdjM_to_quboDict(Q_matrix)
solve_qubo_exact(Q)
```

We find that there are 4 minimum solutions (the 4 Type A solutions). This matches with the expected analytical results.

Though the QUBO solution now works on D-Wave's solver, it needs to be converted to the Ising model for it to run on the D-Wave Quantum Annealer. This can be done using the qubo_to_ising function in D-Wave's toolset, which maps the definitions of binary variables to an Ising model defined on spins (variables with -1, +1 values). The following script solves the Ising model for our formulation.

Mathematically, the transform is: $x^TQx = \text{offset} + s^TJs + h^Ts$. For every linear (diagonal) bias term in Q, $h[i] += 0.5 * Q[i][i]$, while for each couplings, $J[(i,j)] = 0.25 * Q[i][j]$, $h[i] += 0.25 * Q[i][j]$, $h[j] += 0.25 * Q[i][i]$. The offset value is the weighted sum of the linear offset (sum of all diagonal terms in Q), and the quadratic offset (sum of all off-diagonal terms in Q), with the weights as 0.5 and 0.25, respectively. The offset is not important for our case as we want the qubit state of the minimum energy, not the exact value of the minimized energy. This is coded in the `solve_ising_exact` function in `../denovo_009.py`. It can be invoked with the output of `dimod.qubo_to_ising` function as

```
hii, Jij, offset = dimod.qubo_to_ising(Q)
solve_ising_exact(hii,Jij)
```

As expected, we find that there are 4 minimum solutions (the 4 Type A solutions).

```
{'n0t0': +1, 'n0t1': -1, 'n0t2': -1, 'n0t3': -1,
 'n1t0': -1, 'n1t1': +1, 'n1t2': -1, 'n1t3': -1,
 'n2t0': -1, 'n2t1': -1, 'n2t2': +1, 'n2t3': -1,
 'n3t0': -1, 'n3t1': -1, 'n3t2': -1, 'n3t3': +1}

{'n0t0': -1, 'n0t1': +1, 'n0t2': -1, 'n0t3': -1,
 'n1t0': -1, 'n1t1': -1, 'n1t2': +1, 'n1t3': -1,
 'n2t0': -1, 'n2t1': -1, 'n2t2': -1, 'n2t3': +1,
 'n3t0': +1, 'n3t1': -1, 'n3t2': -1, 'n3t3': -1}

{'n0t0': -1, 'n0t1': -1, 'n0t2': +1, 'n0t3': -1,
 'n1t0': -1, 'n1t1': -1, 'n1t2': -1, 'n1t3': +1,
 'n2t0': +1, 'n2t1': -1, 'n2t2': -1, 'n2t3': -1,
 'n3t0': -1, 'n3t1': +1, 'n3t2': -1, 'n3t3': -1}

{'n0t0': -1, 'n0t1': -1, 'n0t2': -1, 'n0t3': +1,
 'n1t0': +1, 'n1t1': -1, 'n1t2': -1, 'n1t3': -1,
 'n2t0': -1, 'n2t1': +1, 'n2t2': -1, 'n2t3': -1,
 'n3t0': -1, 'n3t1': -1, 'n3t2': +1, 'n3t3': -1}
```

D-Wave offers a connection to the cloud to solve an Ising model. The problem needs to be embedded in the connectivity graph of the annealer. It is called a Chimera graph for the D-Wave 2000Q systems. Each of the 16 logical qubits is embedded over multiple qubits on the actual hardware so that each qubit shares a coupling based on the required interaction for the Ising model. The embedding process is a hard problem in itself and heuristics are employed in the D-Wave's embedding function. Thus, with each run, the number of qubits and the longest chain length (i.e. the number of physical qubits encoding a single qubit) might vary, and even fail at times. The embedding process can be separately tested.

This is coded in the `embed_qubo_chimera` function in `../denovo_009.py`. It can be invoked with the output of `dimod.qubo_to_ising` function as

```
embed_qubo_chimera(quboAdjM)
```

After multiple attempts, the best embedding obtained for our example de novo problem uses 60 qubits with a maximum chain length of 5, as shown in Figure 2.16. Each color represents one of the 16 logical qubits.

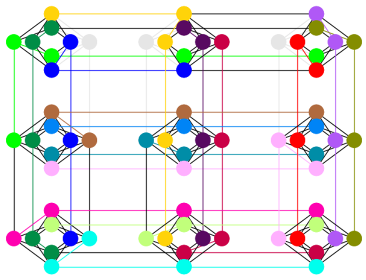


Figure 2.16: Embedding the QUBO in Chimera graph topology for the D-Wave Quantum Annealer. Each color represents one of the 16 logical qubits.

The following code connects to the D-Wave cloud and solves the de novo example. The code to connects to the D-Wave cloud and solve the de novo is in the `solve_ising_dwave` function in `../denovo_009.py`. It can be invoked with the biases and coupling output of the `dimod.qubo_to_ising` function as

```
solve_ising_dwave(hii, Jij)
```

The top 10 (out of 65536) maximum sampled configurations are shown below. It is important to note that, the highest sampled configuration is not the global minima in terms of energy, showing the heuristic nature of the annealer.

```
Maximum Sampled Configurations from D-Wave ===>
    ([-1,-1,-1,+1,-1,-1,-1,-1,+1,-1,-1,-1,-1,-1,+1,-1], -27.92288250, 4562)
    ([-1,-1,+1,+1,-1,-1,-1,-1,+1,-1,-1,-1,+1,-1,-1,-1], -22.70124548, 611)
    ([-1,-1,+1,+1,-1,-1,-1,-1,-1,+1,-1,-1,+1,-1,-1,-1], -26.16476862, 481)
    ([-1,-1,+1,+1,-1,-1,-1,-1,+1,-1,-1,-1,+1,-1,-1,-1], -22.70124548, 474)
    ([-1,-1,-1,+1,-1,-1,-1,-1,-1,+1,-1,-1,-1,-1,+1,-1], -28.08099638, 470)
    ([-1,-1,-1,+1,-1,-1,-1,-1,+1,-1,-1,-1,-1,-1,+1,-1], -27.92288250, 343)
    ([+1,-1,-1,-1,-1,-1,-1,-1,-1,+1,-1,-1,-1,-1,+1,-1], -27.81747324, 295)
    ([-1,-1,-1,+1,-1,-1,-1,-1,+1,-1,-1,-1,-1,-1,+1,-1], -27.92288250, 259)
    ([-1,-1,-1,+1,-1,-1,-1,-1,-1,-1,+1,-1,+1,-1,-1,-1], -27.60665473, 200)
    ([-1,-1,-1,+1,-1,-1,-1,-1,-1,-1,+1,-1,+1,-1,-1,-1], -27.60665473, 187)
```

The top 10 (out of 65536) minimum energy configurations are shown below. The list shows that, though the D-Wave was able to sample two of the four correct solutions, it has not sampled it with a high probability. Also, we find two other solution configurations are missed. Each run of the sampler would be slightly different varying both on environmental errors of the physical qubit system as well as the heuristics of embedding and schedule. Thus, while we were able to find an acceptable solution by the physical system, it might not be practical for larger problems.

```
Minimum Energy Configurations from D-Wave ===>
    ([-1,-1,+1,-1,-1,-1,-1,+1,+1,-1,-1,-1,-1,+1,-1,-1], -30.41886117, 26)
    ([-1,-1,+1,-1,-1,-1,-1,+1,+1,-1,-1,-1,-1,+1,-1,-1], -30.41886117, 2)
    ([-1,-1,+1,-1,-1,-1,-1,+1,+1,-1,-1,-1,-1,+1,-1,-1], -30.41886117, 2)
    ([-1,-1,-1,+1,+1,-1,-1,-1,-1,+1,-1,-1,-1,-1,+1,-1], -30.41886117, 29)
    ([-1,-1,-1,+1,+1,-1,-1,-1,-1,+1,-1,-1,-1,-1,+1,-1], -30.41886117, 1)
    ([-1,-1,-1,+1,+1,-1,-1,-1,-1,+1,-1,-1,-1,-1,+1,-1], -30.41886117, 7)
    ([-1,-1,+1,-1,-1,-1,-1,+1,+1,-1,-1,-1,-1,+1,-1,-1], -30.41886117, 1)
    ([-1,-1,-1,+1,-1,+1,-1,-1,+1,-1,-1,-1,-1,-1,+1,-1], -29.89181489, 23)
    ([+1,-1,-1,-1,-1,-1,+1,-1,-1,-1,-1,+1,-1,+1,-1,-1], -29.89181489, 5)
    ([-1,-1,-1,+1,-1,+1,-1,-1,+1,-1,-1,-1,-1,-1,+1,-1], -29.89181489, 3)
```

A 16 qubit system was required for solving the above problem. When mapping the 16 qubits to a realistic hardware like D-Wave 2000Q, the connectivity of the qubits in the physical topology is important. The embedding process considerably increases the number of required qubits and also the quality of the solution. The highest number of DNA reads that can be solved on a D-Wave 2000Q machine is 9. The amount of qubits needed to solve the problem grows as N^2 and finding embedding for the case with 10 reads will fail in most (if not all) cases. However, the time to solution is independent of the problem size and depends on a heuristic annealing schedule while affecting the quality of the sampled solution.

In classical computation however, the record for exact solutions to the problem, using branch and bound algorithms is 85900 cities TSP [71]. Heuristics like Monte Carlo methods are used for larger inputs. This experiment infers the need for a much enhanced D-Wave system to do practical de novo assembly. Steps in this direction can be either in reducing the errors, having a custom anneal schedule, having more qubits and better connectivity (like the Pegasus architecture of the 5000 qubit model). At the current state of development, we were able to show a simple proof of concept both on the simulator and the quantum annealer on the cloud. The implementation is focused on the correctness of the pipeline design instead of quantifying the time and qubit resource metrics, that need radical improvements for benchmarking with real world datasets. De novo sequencing on quantum annealers needs to be evaluated with each release of improved hardware to reach a quantum advantage in computation over existing high-performance computing systems.

QUBO USING QAOA

Formulating a problem on QAOA involves specifying the ansatz for the cost and driver Hamiltonian. Other optimization hyper-parameters involve the initialization circuit, approximation order (cycles), initial parameters and threshold on classical optimizer iterations/precision. The pseudo-code for the formulation steps in an OpenQL implementation is shown in Listing 2.2.

```
• PQC encoding (e.g. Traveling salesman problem, Maximum cut problem):
  ◇ Create (weighted) graph with networkx
  ◇ Converts graph to weighted–Sum–of–Product–of–Paulis (wsopp). Encoding depends on problem. This is
      the problem/cost Hamiltonian for QUBO/Ising model.
  ◇ Convert graph to ansatz with cost and mixing Hamiltonian as parameterized QASM (ansatz, coefficients,
      angle ids)

• Initialization:
  ◇ Classical optimizer object from SciPy:
    ⋆ name [Default: Nelder–Mead]
    ⋆ convergence function tolerance [Default: 1.0e–6]
    ⋆ iteration limit
  ◇ Reference/initial state quantum circuit [Default: equal superposition, Hadamard on all qubits]
  ◇ Steps (ansatz blocks per iteration) [Default: 1]
  ◇ Parameters:
    ⋆ for cost Hamiltonians (gammas) for each step
    ⋆ for mixing/driving Hamiltonians (betas) for each step
   [Default: random angles in 0 to $2\pi$]
  ◇ Shots (for state tomography measurement aggregate) [Default: 0, QX internal state vector is accessed]

• Invoke QAOA
```

Listing 2.2: Pseudo-code for applying QAOA in OpenQL for optimization.

First, the TSP city-graph is created based on the previous example. The networkx Python package is used to create a directed weighted complete graph based on the pair-wise read edge-weights.

The graph is then converted to the problem Hamiltonian. The problem Hamiltonian is stored as a weighted sum-of-product of Paulis. For n cities (TSP graph nodes), n^2 qubits are required representing the city nodes and the time slots, encoded as:

$$q0 \dots q15 \equiv [n_0 t_0 | n_0 t_1 | n_0 t_2 | n_0 t_3 | n_1 t_0 | n_1 t_1 | \dots | n_3 t_2 | n_3 t_3]$$

On each qubit, there can be either of the 4 Pauli operators, $\{I, X, Y, Z\}$, thus a maximum of 4^{n^2} weighted sum-of-product Pauli terms are possible. This amounts to 4294967296 Pauli terms when $n = 4$ (in our example), thus, we store only the non-zero terms. For TSP optimization, however, only the $\{I, Z\}$ operator is required.

Firstly, each city and each time-slot must be assigned, but not all together. Thus, a term is added with a positive penalty ($w = 100000.0$) for each qubit (the term being a Z operator on the specific qubit and I otherwise). We will abbreviate the sum-of-product of Pauli term notation henceforth by assuming Identity for qubits not mentioned in a Pauli term. Thus:

$$w * \{ZIIIIIIIIIIIIII + IZIIIIIIIIIIIII + \cdots + IIIIIIIIIIIIIIZ\}$$

$$H_C^1 = w * \{Z_0 + Z_1 + Z_2 + \cdots + Z_{15}\} = \sum_{q=0}^{n^2-1} wZ_q$$

Then, for each penalty of co-location (two cities, same time slot), a term with 2 Z operators for the two conflicting qubits is added with a positive penalty weight, and two separate terms for each penalty qubits with a 1 Z are added with a negative penalty.

$$H_C^2 = \sum_{r=0}^{n-1} \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} \left\{ -\frac{w}{2} Z_{in+r} - \frac{w}{2} Z_{jn+r} + \frac{w}{2} Z_{in+r} Z_{jn+r} \right\}$$

Similar terms are added for repetition (two time slots, same city).

$$H_C^3 = \sum_{i=0}^{n-1} \sum_{r=1}^{n-1} \sum_{s=0}^{r-1} \left\{ -\frac{w}{2} Z_{in+r} - \frac{w}{2} Z_{in+s} + \frac{w}{2} Z_{in+r} Z_{in+s} \right\}$$

For each TSP edge, the edge pair is assigned to consecutive time slots. The edge weight is added as a penalty for the 2 Z operator terms, while assigning the individual terms with a negative penalty.

$$H_C^4 = \sum_{i=0}^{n-1} \sum_{\substack{j=0 \\ j \neq i}}^{n-1} \sum_{\substack{r=0 \\ s=(r+1)\%n}}^{n-1} \left\{ -\frac{d_{ij}}{4} Z_{in+r} - \frac{d_{ij}}{4} Z_{jn+s} + \frac{d_{ij}}{4} Z_{in+r} Z_{jn+s} \right\}$$

Thus, these negative penalty terms in the overall equation stand out among the positive penalty terms for all qubits in H_C^1, so that only a valid path is assigned as a solution. The final cost Hamiltonian is $H_C = H_C^1 + H_C^2 + H_C^3 + H_C^4$

Now the ansatz needs to be formed. For the cost Hamiltonian ansatz, for each single Z terms, the Pauli-Z is replaced with a parameterized Z-Rotation gate $R_Z(\gamma)$. The double Z terms between qubits (a, b) are replaced with a $CNOT_{ab} R_Z^b(\gamma) CNOT_{ab}$ (where b is the target qubit). For the mixing Hamiltonian, a $R_X(\beta)$ is used on all qubit (or $HR_Z(\beta)H$).

The reference state is set to an equal superposition (Hadamard over all qubits). The cost and mixing Hamiltonian ansatz is alternated for the set number of steps (1 in our case). This forms the parametric circuit for the quantum computer. Along with this, the random initial parameters vector (Γ, B) is passed to the classical optimizer wrapper for the QAOA. The QAOA is run as explained in listing 2.1.

We simulated our algorithm on the QX Simulator to validate the results. To speed up the simulation, we access the internal state vector from the QX Simulator instead of applying the state tomographic trials. However, optimizing the parameters on 16 qubits for single iteration over a single QAOA step proved to be cumbersome, reaching the limit of the working memory for most runs. While we obtain good results for the Max-Cut problem (which requires as many qubits as nodes in the graph), the n^2 qubit space for TSP is costly [72] for simulation. Similarly, most online tools only offer examples for the trivial case of an undirected triangle graph (where only one Hamiltonian cycle is possible). We used the Nelder-Mead, Powell and BFGS optimizers in SciPy. We found that the optimizers are able to explore only a small space near the initial guess before settling at a suboptimal solution. This is shown in Figure 2.17, where the dotted vertical green lines indicate the 4 optimal type A solution states out of the $2^{16} = 65536$ basis states (represented on the x-axis). The experiment uses a QAOA depth of 1, a random angle initialization, an equal superposed initial state (Hadamard on all qubits) and 40 reruns for the QAOA optimizer. When the initial guess is bad, the highest probability states (high blue circles) are not close to the optimal lines, whereas for good initial conditions case (red pluses), the optimizer eventually reaches solutions that are quite close to the optimal. The optimal solutions lie in near vicinity of the found solutions (high red pluses) and can be subsequently explored via an exhaustive search near these sub-optima suggested by QAOA. QAOA is able to improve 2 out of the 4 solutions of Type A. Development of new classical optimizers [73, 74] and their hyper-parameter settings [75] for HQC algorithms is a research field of its own, which is not the focus of this paper. Thus, while we were able to formulate a generic de novo assembly problem to QAOA, we were not able to obtain satisfactory results from the simulation. This motivates the need for both faster simulation and the access to better NISQ devices where this entire pipeline can be executed and benchmarked - a common challenge for quantum computing today.

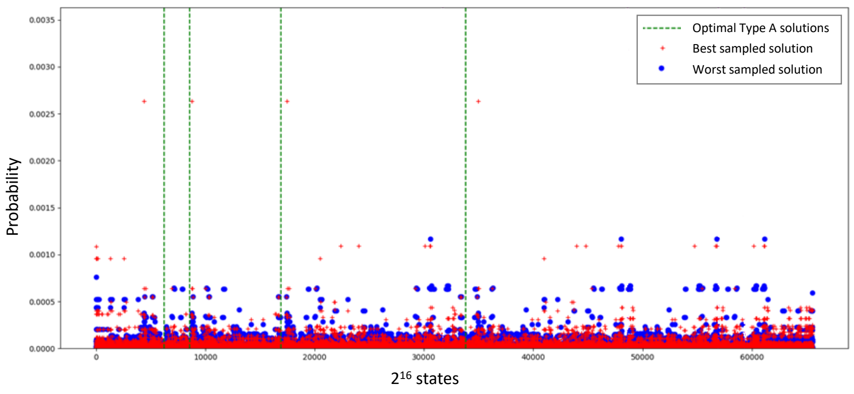


Figure 2.17: Result of simulating QuASeR example using QAOA on the QX Simulator

QAOA, though promising for exhibiting quantum supremacy does not imply that it will be able to outperform classical algorithms on important combinatorial optimization problems such as Constraint Satisfaction Problems. Current implementations of QAOA are subject to a gate fidelity limitation, where the potential advantages of larger values of the parameter p in QAOA applications are likely to be countered by a decrease in solution accuracy.

2.5. CONCLUSIONS AND OUTLOOK

The research presented in this chapter is motivated by the bottleneck of DNA sequence reconstruction in genomics, and explores how quantum acceleration can be applied in this domain. Quantum-accelerated genome sequence reconstruction was not studied before. However, it was found that, this dearth of previous work is not because it is not suited for quantum acceleration. The field of quantum computing is young and under rapid development. This research was among the first few to explore the applications of quantum computing in bioinformatics.

There exists a considerable technological readiness gap between the resource requirements of realistic quantum algorithms and available physical quantum processors. Thus, most developments in QC algorithms are agnostic to hardware developments and focus on theoretical proofs of advantages in terms of computational resources. On the other end of the spectrum, there are trivial algorithms that are used to demonstrate and benchmark the computing capabilities of quantum hardware platforms and characterizing the instrumentation. The intent of this research instead was to implement proof of concept simulation of application driven quantum algorithm, thereby, to decompose the mathematical formulations and oracles in terms of quantum logic gates that can be executed on a classical quantum computing simulator.

Genome sequence reconstruction can be done in two ways: reference alignment (ab initio) or read assembly (de novo). The big data problem in reference alignment can be readily mapped to an unstructured search problem, while read assembly is based on graph traversal optimization. Since the Grover search is a well developed quantum algorithm primitive, the reference alignment approach was first explored. In § 2.3 of this chapter, the proposed QiBAM algorithm is discussed. This is the first time a quantum pattern matching algorithm is specifically designed, keeping in mind genomic sequences. The idea of associative memory was extended to an indexed directory of DNA sequences spliced from the reference genome. In addition to taking into account the DNA alphabet, since reads can contain errors, a distributed query for approximate matching was designed. This was applied over the superposition of a quantum state, thereby storing an exponential number of patterns. A constant oracle was designed based on minimizing the Hamming distances. This eliminates the bottleneck of compiling the query differently for every short read at run-time. The associated index in the reference was retrieved, instead of the corrected query by entangling the index with the sequence database. The algorithm was implemented and verified in the OpenQL environment with the QX Simulator as the backend. The algorithm was generalized to a generic quantum data structure for multi-dimensional search.

The simulation capability of QiBAM reaches its limit for very short read lengths. The current noisy intermediate-scale quantum era is also not favorable in terms of qubits multiplicity, topological connectivity, control restrictions and error rates. Thus, real datasets cannot be integrated in the algorithm pipeline in the near term. Classical state of the art algorithms rely on many different heuristics to further simplify the alignment, which are not readily portable on quantum logic. In consultation with some industrial connections, it was understood that the genomics industry is moving towards reference-bias free methods. The third generation single molecule sequencer technology with longer reads over next-generation sequencers might provide better sequence reconstruction via the de novo methods. Given the receding timeline of fault-tolerant quantum computing and the advent of better single molecule sequencers, alternate genome sequence reconstruction techniques were targeted. Thus, having developed one of the world's first quantum-accelerated genomics algorithms, we decided to explore the alternate approach of de novo assembly in further research. This approach, called QuASeR, targets the overlap-layout-consensus method for acceleration on a quantum computing platform. The quantum kernel is formulated for both a gate-based quantum system as well as a quantum annealer. The main difference between QuASeR and QiBAM is the shift in focus from FTQC era to the NISQ technology era. QiBAM is based on the primitive of Grover's search, which, though provably optimal, is costly both in terms of the quantum circuit depth (time, number of quantum gates) and width (space, number of qubits). The current trend is to focus on heuristic approaches in quantum algorithms. These algorithms are hybrid in nature, where the quantum accelerator executes a low-depth parameterized circuit that prepares and measures a quantum state. A classical wrapper is an optimizer that selects these parameters (typically angles for rotation gates) for each trial run of the quantum accelerator, till the quantum circuit is evolved to replicate the behavior of minimizing a cost function. The cost function is designed to encode the computation required for the target application. These heuristics come under the umbrella of variational hybrid quantum-classical approach (VHQCA). The corner-stone of this research was the quantum approximate optimization algorithm (QAOA), an universal algorithm for approximating the solution for a discrete combinatorial optimization problem. Since these pervasive societal/industrial problems are mostly NP-hard (thus, scales exponentially even on quantum), a fast yet good heuristic solution is often desirable for pragmatic cases. The problem is typically modeled as a quadratic unconstrained binary optimization (QUBO) formulation.

In § 2.4 the formulation of a de novo assembly DNA sequence reconstruction algorithm is presented. The required technical background for formulating the de novo sequencing problem (i.e. QUBO, TSP, and Hamiltonians) is introduced with simple examples to target both the genomics research community and quantum application developers. A proof of concept de novo sequence assembly is mapped to TSP and then to a QUBO. This is firstly solved on the D-Wave simulator and D-Wave Quantum Annealer. All 4 correct results are obtained on the simulator while only 2 of the solutions are sampled on the Quantum Annealer (though with less probability). The connectivity topology of the D-Wave architecture limits embedding larger problem instances. The variational algorithm approach for gate-based quantum computing is introduced for solving optimization problems using QAOA. This algorithm performs two optimization steps, one executed on a quantum circuit and another on a classical computer. The proposed de novo algorithm is solved using QAOA, and then simulated on the QX simulator. Simulation showed that the results are heavily dependent on the exploratory capabilities of the classical optimizer, which is an open research question in the community. Also, the ansatz for the variational circuit is a heuristic which needs to be designed to fit our purpose. In this respect, a learning approach might prove promising. A gate-based quantum computer is not targeted as the coherence time, connectivity topology and number of qubits prevents any meaningful result at the current state of available quantum hardware. It was found that similar limitations apply for de novo as were for ab initio, where real datasets are very far away from the simulation capabilities of QX/Ocean and NISQ hardware. Also, it is not trivial to introduce data-specific heuristics in QUBO. The biggest limitation however is that both these methods (QAOA and QA) are quantum heuristics, and to further optimize these algorithms experiments on real hardware are imperative.

Beside the presented research, quantum Hamiltonian complexity (QHC) was briefly studied. It is the theoretical study of the properties of the Hamiltonian formulation for finding the energy spectra of a physical system or an optimization landscape. It was studied in the context of the cost Hamiltonians for QAOA to compare, (i) what can be efficiently simulated classically, (ii) what is submissive to quantum accelerations and, (iii) what is tough even for quantum computation. QHC's mathematical approach deals mainly with the cases of exact solutions, which is not practical due to our dependence on heuristics for NP-hard problems. Fujitsu's digital annealer (DA) as an alternative to gate-based quantum computing were explored. While the final conclusion was that annealing in general is not expressive enough to formulate the requirements of the logic; this study gave valuable experience in formulating QUBO and Hamiltonians. Xanadu's PennyLane toolset was also explored for hybrid quantum-classical programming. While this provides an automatic quantum circuit differentiation tool for gradient-based QAOA optimizer, the rigid structure of the programming currently outweigh its advantage over OpenQL.

The research presented in this chapter is the first exploration towards a roadmap project [6] undertaken within the Quantum Computer Architecture team at the Delft University of Technology, to design a 'full-stack quantum accelerator architecture, domain-specific for genomics'. The work from this chapter is currently being refined as an industrial pipeline in collaboration with the quantum startup QBee B.V. This research was conducted over a period of 2 years (Nov., 2017 to May, 2018, and Nov., 2018 to Oct., 2019). It helped us in revising our understanding of the timeline of quantum advantage, specifically for bioinformatics applications. With the new projection, our genomics partners predict that instead of genome reconstruction, genome data analysis will most likely be the bottleneck beyond a 5+ year timeline when large scale QC will likely be available. Thus, this informs the direction for further research presented in the following chapters.

3 QUANTUM AUTOMATA FOR ALGORITHMIC INFORMATION

The research on quantum-accelerated genome sequence reconstruction discerned the implication of algorithm development based on current quantum processors and quantum computing simulators. The aim of further research is to look beyond near-term approaches like variational algorithms that can be applied for genomics. Given the timeline of the development of large-scale quantum accelerators, genome sequence analysis is a more promising target application. Inferring algorithmic structure in data is essential for discovering causal generative models. Such a framework will have significant advantage for various genomics applications.

We realized that at the current technology readiness level, the quantum approach should focus on implementing the algorithmic primitive instead of the full application pipeline. In this chapter, we present the proposed quantum computing framework using the circuit model for estimating mathematical objects in the context of algorithmic information. The canonical computation model of the Turing machine is restricted in time and space resources, to make the target mathematical objects computable under realistic assumptions. The universal prior distribution for the automata is obtained as a quantum superposition, which is further conditioned to estimate them. 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's search that cannot be dequantized. Experimental

This chapter is based on the following:

- **Sarkar, A.**, Al-Ars, Z., & Bertels, K. (2020). Quantum circuit design for universal distribution using a superposition of classical automata. *arXiv preprint arXiv:2006.00987.*
- **Sarkar, A.**, Al-Ars, Z., & Bertels, K. (2021). Estimating algorithmic information using quantum computing for genomics applications. *Applied Sciences, 11*(6), 2696.

use cases for quantum acceleration are developed for self-replicating programs and algorithmic complexity of short strings. This is the first time experimental algorithmic information theory is implemented using quantum computation.

3.1. Use case motivation

The phenomenal success of data-driven approaches like deep learning has ushered automation in many spheres of human society over the last decade. However, these approaches, which are based on black-box optimization, provide limited insights on the causal generative mechanism underlying the set of observations about the physical process under study. These limitations in explainability are increasingly becoming crucial with automation in mission critical sectors like healthcare.

In contrast, symbolic approaches have been successfully used to model, study and understand the causal relationships in natural phenomena and datasets. The algorithmic information theoretic approach [76] 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.

The set of transformations a computation model can undergo and the resulting space of outputs are central to understanding the causal structure of the physical phenomena we intend to model. Except for the trivial cases, this remains intractable on classical computers since the space of all possible transformations grows exponentially with the number of states and symbols of the automata. The uncomputable nature of these mathematical objects is approximated in practice by restricting the resources available to the computational models, such as time/cycles and space/memory. Even with such restrictions, only relatively simple automata have been explored using supercomputing clusters [77]. These limited results have already found various applications in genomics, psychology, network science, image processing, etc. In biological systems, a better understanding of the algorithmic structures in DNA sequences [78] and cellular dynamics [79] would greatly advance domains such as personalized medicine.

We explore the possibility of accelerating this technique on a (gate-based) circuit model of quantum computation. In this research, we propose a detailed circuit implementation that allows encoding a superposition of classical programs and evaluating their evolution after a predetermined number of cycles. Thereafter, we present a full experimental framework for using this approach for two use cases, i.e. for querying the subspace of quines, and of estimating the algorithmic complexity of short strings. Specifically in genomics, it finds applications in meta-biology, phylogenetic tree analysis, protein-protein interaction mapping and synthetic biology.

3.2. Algorithmic model

Defining an algorithmic process via symbolic manipulations requires a computation model. The Turing machine model of computation is the cornerstone of theoretical computer science. This simple mechanistic automata has the expressive power to capture any algorithmic process. Thus, it can be used for generic modeling and hypothesis comparison across various scientific disciplines. In this research, we follow the algorithmic information theoretic approach of enumerating the distribution of all generating automata (e.g., Turing machines). The set of transformations a computation model can undergo and the resulting space of outputs is central to understanding the causal structure of the physical phenomena we intend to model.

In this research, we develop techniques to accelerate this application on a (gate-based) circuit model of quantum computation. The quantum computation model provides distinctive advantages for specific algorithms using the laws of quantum mechanics. Recently, a generic (gate-based) circuit model of quantum Turing machine was proposed [80] based on cellular automata. In this research, we propose an alternative model from a mechanistic perspective, with a detailed circuit implementation with realistic assumptions on runtime and qubit resources for the tape memory. We do away with the homogeneous local structure resulting in execution of many inactive unitary transforms, thereby improving the total number of executed operations. The proposed model intuitively allows encoding a superposition of classical programs and evaluating their evolution after a predetermined number of cycles. We present the exact scalable circuit using standard quantum gates required to simulate a superposition of this resource-bounded stored-program automata. With the recent thrust in the realization of quantum computing hardware, this is a promising direction with multifaceted application that can extend the applications of approximating algorithmic mathematical objects by enumerating automata configurations. The availability of better quantum processors would allow this algorithm to be readily ported on a quantum accelerator with a quantum computing stack [6]. 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, and resource complexity estimation techniques. The necessary background for these is presented in this section. In § 3.3, 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. This formalism is applied for specific applications in algorithmic information theory (AIT). The background for AIT is presented in § 3.4. Experimental use cases for self-replicating programs and algorithmic complexity, and their applications in genomics using the developed framework are presented in § 3.5.

3.2.1. TURING MACHINE MODEL

The Turing machine (TM) is the canonical model of computation invented by Alan Turing for proving the uncomputability of the Entscheidungsproblem. A TM manipulates symbols on an infinite memory strip of tape divided into discrete cells according to a table of transition rules. 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 3.1, 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.

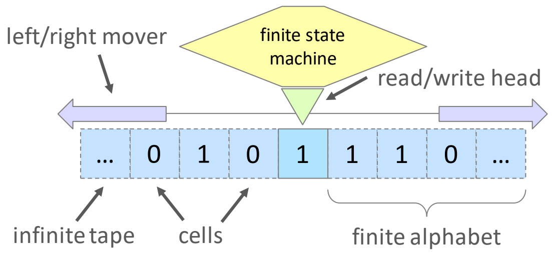


Figure 3.1: Computational model of a Turing machine.

Each model of computation defines a set of inputs that are accepted by that automata. For the Turing machine, this subset (of all possible strings over the alphabet of the language) that can be enumerated (outputs the string) is called recursively enumerable. Recursively enumerable languages (also called, Turing recognizable/acceptable, semi/partially decidable) forms the type-0 in the Chomsky hierarchy of formal languages, as discussed in the following section.

3.2.2. VARIANTS OF THE TM MODEL

While Turing machines can express arbitrary mechanical computations, their minimalist design makes them unsuitable for algorithm design for computation in practice. Various modifications of the TM results in equivalent computation power, as captured by the Church-Turing thesis. Here, equivalent refers to being within polynomial translation overhead in time or memory resources. Thus, in practice they might compute faster, use less memory, or have smaller instruction set, but they cannot compute more mathematical functions. These modified models of computation, like lambda calculus, Post machine, cyclic-tag system, offers different perspective in development of computer hardware and software. Extending the TM tape to multiple dimensions is also equivalent to a 1-dimensional tape (or, only 1-way infinite tape).

In this section, we discuss some variants of the Turing machine model, as listed in Table 3.1, that are relevant for further ideas developed in this chapter. These variants can also be applied to the corresponding language instead of the automata. Turing completeness is the ability for a system of instructions to simulate a Turing machine. A Turing complete programming language is thus theoretically capable of expressing all tasks that can be accomplished by computers. Nearly all programming languages are Turing complete if the limitations of finite memory are ignored.

A universal Turing machine (UTM) simulates an arbitrary Turing machine on an arbitrary input. Every TM can be assigned a number, called the machine's description number, similar to Gödel numbers, encoding the FSM as a list of transitions. The existence of

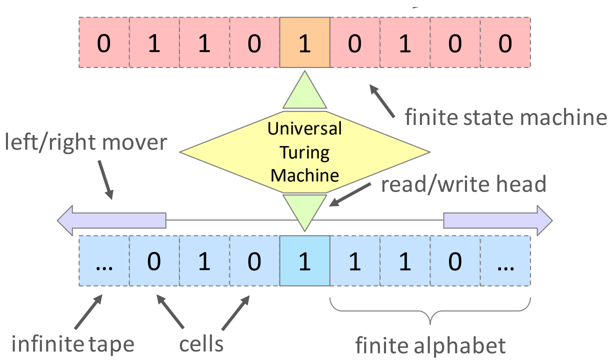


Figure 3.2: Computational model of a universal Turing machine.

this direct correspondence between natural numbers and TM implies that the set of all Turing machines (or programs of a fixed size) is denumerable. 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 3.2. This is the idea behind the stored-program von Neumann architecture. This is the first variation to automata models producing the automata variants in Table 3.1.

The sequential tape movement in TM/UTM makes them unsuitable for computation in practice (e.g. a binary search is really slow on a TM). This is alleviated by extending the capability of the memory to access any indexed tape cell. These variants of TM and UTM models are called random-access machine (RAM) and random-access stored-program (RASP) model respectively. This is the second variation to automata models in Table 3.1.

Multiple FSMs can act based on the same tape data, allowing a shared memory for a multi-core processing model. This modification is called the parallel TM (PTM) model and equivalently the PUTM, PRAM and PRASP, as the third variation to TM models in Table 3.1.

3.2.3. CHOMSKY HIERARCHY OF AUTOMATA AND LANGUAGES

For real implementations of an automata an infinite tape is not possible. Thus computational models restrict the tape features in various ways [81, 82]. These result in a reduced level in the Chomsky hierarchy of formal languages and the corresponding automata model that accepts strings from the respective grammar.

- Type-0: Recursively enumerable language; Turing machine (TM)
- Type-1: Context-sensitive language; Linear bounded automata (LBA)
- Type-2: Context-free language; Pushdown automaton (PDA)
- Type-3: Regular languages; Finite state machine (FSM)

Each higher level (the highest being type-0) can always simulate the lower level. Restrictions in memory make the other levels computationally less capable than a TM in terms of the language, (i.e. the set of string patterns called the grammar) it can recognize. For example, an FSM cannot determine if an input string has the structure $a^i b^i$, like *aabb*, *aaaabbbb*. At type-0, universality is reached, thus, everything that is com-

Table 3.1: Variants of the Turing machine model for deterministic, non-deterministic and quantum automata classes. Our implementation of QPULBA (in yellow) captures the computing capabilities of 27 (in blue) out of 51 automata models (of type 3, 2, 1) that is realistically implementable on physical hardware.

		Automata class			Features					
	Type	Deterministic	Non-deterministic	Quantum	Stored program	Random memory access	Parallel	Infinite memory	Movable tape head	Memory based
1	3	FSM	NFSM	QFSM	N	N	N	N	N	N
2	2	PDA	NPDA	QPDA	N	N	N	N	N	Y
3		UPDA	NUPDA	QUPDA	Y	N	N	N	N	Y
4		PDRAA	NPDRAA	QPDRAA	N	Y	N	N	N	Y
5		PDRASP	NPDRASP	QPDRASP	Y	Y	N	N	N	Y
6		PPDA	NPPDA	QPPDA	N	N	Y	N	N	Y
7		PUPDA	NPUPDA	QPUPDA	Y	N	Y	N	N	Y
8		PDRAA	NPDRAA	QPDRAA	N	Y	Y	N	N	Y
9		PPDRASP	NPPDRASP	QPPDRASP	Y	Y	Y	N	N	Y
10	1	LBA	NLBA	QLBA	N	N	N	N	Y	Y
11		ULBA	NULBA	QULBA	Y	N	N	N	Y	Y
12		LBRAM	NLBRAM	QLBRAM	N	Y	N	N	Y	Y
13		LBRASP	NLBRASP	QLBRASP	Y	Y	N	N	Y	Y
14		PLBA	NPLBA	QPLBA	N	N	Y	N	Y	Y
15		PULBA	NPULBA	QPULBA	Y	N	Y	N	Y	Y
16		PLBRAM	NPLBRAM	QPLBRAM	N	Y	Y	N	Y	Y
17		PLBRASP	NPLBRASP	QPLBRASP	Y	Y	Y	N	Y	Y
18	0	TM	NTM	QTM	N	N	N	Y	Y	Y
19		UTM	NUTM	QUTM	Y	N	N	Y	Y	Y
20		RAM	NRAM	QRAM	N	Y	N	Y	Y	Y
21		RASP	NRASP	QRASP	Y	Y	N	Y	Y	Y
22		PTM	NPTM	QPTM	N	N	Y	Y	Y	Y
23		PUTM	NPUTM	QPUTM	Y	N	Y	Y	Y	Y
24		PRAM	NPRAM	QPRAM	N	Y	Y	Y	Y	Y
25		PRASP	NPRASP	QPRASP	Y	Y	Y	Y	Y	Y

putable can be mapped to an algorithmic process on the TM. This includes quantum computation as well, as it can be simulated on a classical TM (albeit using worst-case exponential resources).

A linear bounded automata (LBA) has a finite memory, extending the same modifications as for the TM, as listed as the fourth modification to TM models in Table 3.1 forming the type-1 in the Chomsky hierarchy. Most real-world computers can be best modeled as the Parallel Linear Bounded Random Access Stored Program (PLBRASP) automata. This allows multiple concurrent processors with a shared random access finite sized memory. The program as well as the data are accessed from the memory in the von Neumann architecture.

In this work, we will consider the Parallel Universal Linear Bounded Automata (PULBA) model which is the sequential memory access variant of the PLBRASP. Thus, we consider multiple stored programs running in parallel while sharing a common limited work memory and output interface.

3.2.4. CELLULAR AUTOMATA MODEL

Often for physical systems, a cellular automata model is preferred as an alternative to Turing machines, specifically when there is a homogeneous local interaction, e.g. in geographic spacial planning and ecological systems. Cellular automata is a discrete model of computation consisting of a regular grid of cells in any finite dimension (or type of tessellation). Each cell at any step in time is in one of a finite number of states, (such as on and off). For each cell, a set of cells called its neighborhood is defined relative to the specified cell. The next step in time is created according to some fixed rule (typically same over the entire space) that determines the new state of each cell in terms of the current state of the cell and the states of the cells in its neighborhood. The whole grid typically updates simultaneously. The model with 1-dimensional tape over 2 symbols and a neighborhood size of 3 (1 cell in either size along with the current cell), is called elementary cellular automata (ECA). These were studied extensively revealing interesting features like the Rule 110 (one of the 256 possible rules) capable of universal computation [83].

Solid-state implementation [84] of cellular automata involves expressing the update rule in terms of classical logic gates (e.g. AND, OR, NOT, NAND). For each cell, the cell and its two immediate neighbors are read, their values entered into the transition function, and the cell updated with the new value for the subsequent generation. The transition function can be represented as a truth table, a logic function or a logic circuit. The result from this function is then written to the corresponding cell in a duplicate array. Transition functions are being calculated and written sequentially to a separate register. Once a generation is complete, the original array is cleared, and the results of rule applications to cells are written to the original register, like a ping-pong buffer.

A cellular automata can simulate a TM by storing a activation bit indicating the position of the tape head and the current state for each cell. The transition function of the TM is encoded as the update rule depending only on the current cell. Thus, it is like PTM with identical FSM, with only one of them active at each cycle. This model is wasteful in terms of logic gate operations as only one of the cells is active yet all the other logic gates need to be executed none the less.

3.2.5. COMPUTATION COMPLEXITY CLASSES

Thus far we discussed computability, i.e. limits on the feasibility of any algorithms based on the language and automata models that accept strings based on the grammar, and automata theory, i.e. the formal model of computation required for a problem. Complexity theory bridges the gap between practical algorithms running on computing hardware and the hierarchy of languages in computability theory. It grades algorithms for a specific automata based on the time and space resources. The complexity of algorithms has been classified into a multitude of classes [85]. The boundaries and relationships between these classes [86] are sometimes not proven but are based on current knowledge and popular conjectures in the scientific community. Here we summarize some theoretical results of possible advantages in quantum approaches for various complexity classes, including approximating solutions to uncomputable problems.

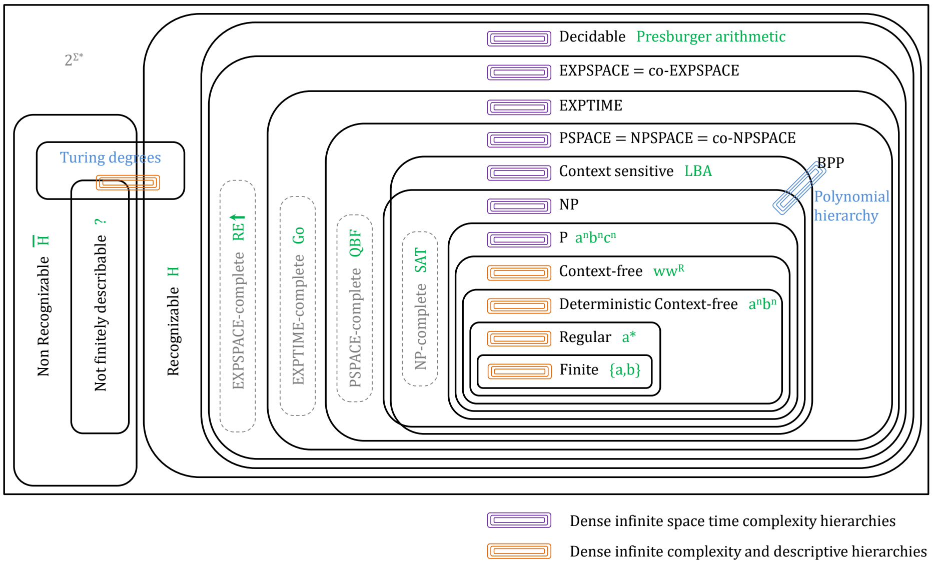

2^{Σ^*}
Decidable Presburger arithmetic
EXPSPACE = co-EXPSPACE
EXPTIME
PSPACE = NPSPACE = co-NPSPACE
Context sensitive LBA
BPP
NP
Polynomial hierarchy
P $a^nb^nc^n$
Context-free ww^R
Deterministic Context-free a^nb^n
Regular a^*
Finite {a,b}
Turing degrees
Non Recognizable $\overline{H}$
Not finitely describable ?
Recognizable H
EXPSPACE-complete RE↑
EXPTIME-complete Go
PSPACE-complete QBF
NP-complete SAT
Dense infinite space time complexity hierarchies
Dense infinite complexity and descriptive hierarchies

Figure 3.3: Extended Chomsky hierarchy and some computational complexity classes (based on [87]). Examples are in green.

The most commonly referred complexity classes are of P and NP. Their relation to computability levels as shown in Figure 3.3 and the quantum complexity classes are important for efficient algorithm development. Polynomial time (PTIME or P for short) refers to the class of algorithms that are efficiently solvable (or tractable) by a deterministic Turing machine in an amount of time that is polynomial in the size of the input problem. Non-deterministic polynomial time (NTIME or NP for short) refers to the set of problems that are tractable by a non-deterministic Turing machine (NTM) in polynomial time. P can easily be reasoned to be a subset of NP (might not be a proper subset). Non-deterministic variants of the relevant automata are listed in Table 3.1. Whether an algorithm that can be efficiently checked for correctness can also be efficiently solved is an open question [88] (one of the Millennium problems). Another important concept here is of NP-completeness, which is the set of problems in NP such that every other problem in NP can be transformed (or reduced) to it in polynomial time.

Often the access to a random-number source widens the class of feasible problems for a computer using probabilistic computation. It is not well understood why randomness acts as a computational resource and if all algorithms can be derandomized without significantly increasing their running time or space. Bounded-error probabilistic polynomial-time (BPP) class involves decision problems solvable by an NTM such that the true answers are achieved with $\geq \frac{2}{3}$ of the computation paths, while the false answers are achieved with $\leq \frac{1}{3}$ paths. Using Chernoff bound, the constant $\frac{1}{3}$ can be reduced arbitrarily on repetition. These constants $\left(\frac{2}{3},\frac{1}{3}\right)$ for BPP, when modified to $(1,0)$ gives zero-error probabilistic polynomial-time (ZPP), $\left(\frac{1}{2},\frac{1}{2}\right)$ gives the probabilistic polynomial-time (PP) class and $\left(\frac{1}{2},0\right)$ gives the randomized polynomial-time (RP) class. Adding post-selection capabilities to these classes gives the class BPP_{path}.

Bounded-error quantum polynomial-time (BQP) is the class of decision problems solvable by a quantum circuit whose length and number of qubits scale polynomially with respect to the instance size. Like its classical counterpart BPP, the error probability is bounded by at most $\frac{1}{3}$ for all instances. Similarly, the quantum equivalent of ZPP and BPP_{path} are exact quantum polynomial-time (EQP) and PostBQP (shown equal to PP). There are also many related classes based on verifiable proof systems (e.g. MA, IP, AM) and their quantum counterparts (QMA, QIP, QAM).

To compare the efficiency of quantum algorithms, it is crucial to compare them with the current best algorithm on classical computers and their complexity. The relationships of quantum computation complexity classes with the class hierarchy are not fully understood and are riddled with many open conjectures. Since P is a subset of BQP, it is beneficial to study algorithms that fall outside P, but in BQP. Also, it is known that quantum algorithms cannot solve NP-complete problems [8] in polynomial time, but Grover's search can offer a quadratic speedup for unstructured problems when good heuristics are not known. Such polynomial speedup can boost these problems into the domain of practicality.

Based on the current physical theory, quantum computers are the most general kind of computers physically allowed [9]. More exotic computers would require some refinement of the laws of physics (like non-linearity, allowing faster than light information transfer, violating uncertainty principle, or using closed time-like curves). Quantum algorithms however do not allow super-Turing computation (e.g. solving the halting problem) thus placing them at the same Turing degree in the arithmetic hierarchy.

3.2.6. ASYMPTOTIC COMPLEXITY

In an implementation of an algorithm we need to estimate the scaling of space/qubit and time/gate resources. The Bachmann–Landau notation (or asymptotic notation) is used to describe the limiting behavior of a function with the domain tending towards a particular value (often infinity). The big O notation is used to classify algorithms according to their running time or space requirements growth rate with input size. A description of a function in terms of big O notation provides an upper bound on the growth rate of the function.

Formally, for real/complex valued functions f and g defined on some unbounded subset of the real positive numbers, $f(x) = O(g(x))$ as $x \to \infty$ iff $\forall$ sufficiently large x $\exists\ M \in \Re$ and $x_0 \in \Re$ such that $|f(x)| \leq Mg(x)\ \forall\ x > x_0$

The O notation for a function f is derived by 2 simplification rules: (i) if $f(x)$ is a sum of several terms, term with largest growth rate is kept, (ii) if $f(x)$ is a product of several factors, constants are omitted. For example, $f(x) = 4x^4 + 2x^3 + 100$ has order $O(x^4)$. Infinite asymptotic analysis often looks over the lower order terms and constants, which might be the deciding factor for practical applications. Even for exponential problems in $O(n^n)$ versus constant time $O(1)$, there is a cross-over of applicability, where the preference shifts. It is important to estimate where the problem of interest lie for stricter comparison among algorithms.

3.3. QUANTUM AUTOMATA USING QUANTUM CIRCUITS

A quantum mechanical model of Turing machines was described by Paul Benioff using Hamiltonian models [89, 90]. A computationally equivalent model using quantum gates (inspired by classical Boolean logic gates) in a quantum circuit was proposed by David Deutsch [12, 13, 91]. Due to the more intuitive nature of the circuit model it has become the standard for quantum algorithm development. A generalized quantum Turing machine (GQTM) [92], (which contains QTM as a special case and includes non-unitary dynamics e.g. irreversible transition functions as well) allow the representation of quantum measurements [93]. From a quantum computer architecture perspective, QTM was compared [94] to quantum random-access machine (QRAM) and quantum random-access stored-program (QRASP) model, proving their equivalence in bounded-error polynomial-time formulations using the Solvay-Kitaev theorem. The QRASP considers that a single program is stored in classical registers, and thus treated as classical data, while the work memory can be in a superposition. A QRASP model of a quantum computer can encode a program as quantum data, consequently generalizing the QRASP model to parallel quantum random-access stored-program machines (PQRASP) allowing a superposition of programs [95].

Here we review the circuit implementations of quantum finite automata (QFA) [96, 97], quantum equivalent automata models [98], and more extensively QTM [80] which inspires our circuit architecture of a quantum parallel universal linear bounded automata (QPULBA). Like its classical counterpart, a quantum Turing machine (QTM) is an abstract model capturing the power of quantum mechanical process for computation. Any quantum algorithm can be expressed formally as a particular QTM. QTM generalizes the classical Turing machine such that the internal states of a classical TM are replaced by pure or mixed states in a Hilbert space. The transition function is replaced by a collection of unitary matrices that map the Hilbert space to itself.

3.3.1. QUANTUM TURING MACHINES

A classical TM is extended to the complex vector space for a QTM. For a three-tape QTM (one tape holding the input, a second tape holding intermediate calculation results, and a third tape holding output):

- The set of states is replaced by a Hilbert space.
- The tape alphabet symbols are replaced by a Hilbert space (usually a different Hilbert space than the set of states).
- The blank symbol corresponds to the zero-vector.
- The input and output symbols are usually taken as a discrete set, as in the classical system; thus, neither the input nor output to a quantum machine need be a quantum system itself.
- The transition function is a generalization of a transition monoid. It is a collection of unitary matrices that are automorphisms of the Hilbert space.
- The initial state may be either a mixed state or a pure state.
- The set of final or accepting states is a subspace of the Hilbert space.

It is important to realize that this is merely a sketch than a formal definition of a quantum Turing machine. Some important details like how often a measurement is performed are not explicitly defined. This is circumvented by formal proofs showing equivalence with an oblivious QTM, whose running time is a function of the size of the input (independent of the structure of the input). However, how to practically translate that to a priori knowledge of the number of steps the computation needs to be executed before collapsing the superposition still needs further exploration [99, 100].

3.3.2. CELLULAR AUTOMATA INSPIRED QTM CIRCUIT

QTM was developed in the 1990s as a formal model to represent quantum computation. Recently, [80, 98] revived this direction of research by presenting quantum circuit formulations and discussing applications of simulating quantum automata.

The approach taken in [80] resembles a (quantum) cellular automata in its homogeneous circuit template and local unitary operations. The no-cloning principle prevents a direct translation of the sequentially updating classical cellular automata architecture to quantum circuits. This is because, when the state is sequentially updated, we lose the neighbor information to update the consecutive cell. The circuit architecture of this QTM is inspired by the solid-state implementation of the computational equivalent model to a TM of a 1-dimensional elementary cellular automata. The local simultaneous update is achieved by maintaining a local one-hot marker denoting the presence/absence of the tape head at the location activates a specific (or superposition of) computation path in the QTM. It defines a fixed unitary G which encodes the rule, or the transition function and interleaved in a particular way. For the quantum version, the inputs are qubits, so they can be in a superposition of states, allowing simultaneous evolution of various inputs (initial Turing tapes).

By this approach, $t \geq n$ steps of a QTM on an input of length n can be simulated by a uniformly generated family of quantum circuits linear in t. However, not all transition functions describe valid quantum Turing machines as it is imperative that the global evolution is unitary on the Hilbert space. Any QTM that exhibits a local causal behavior can be mapped to the transition function G proposed in [80]. Since the model is universal, there exists a transition function that will in effect read a stored program and the input from the tape as qubits and perform the computation entirely by local operations. However, it is not immediately clear how to construct such a unitary for an arbitrary transition function. Moreover, modeling the stored-program model on this local interaction architecture would incur a high cost in the number of operations to affect the tape, as at each step a TM head moves only by one step.

3.3.3. Proposed computation model

In this section, we propose the formulation of our computation model that we translate to the quantum circuit. This complements the research as presented in [80] from a different perspective. The major considerations to make the quantum circuit practically implementable and tractable in resources are discussed.

A mechanistic perspective

The intricacy in applying the cellular automata based QTM to arbitrary algorithms inspires our research to look at more intuitive model with structural similarity to a TM. We do away with the local transition function applied in parallel based on activation qubits. Our quantum circuit provides a mechanistic model of a Linear Bounded Automata, while allowing both the inputs (tape memory) and the program (transition function) to be in superposition. Thus, in effect it provides a scalable quantum circuit implementation for the QPULBA model of computation.

The superposition of programs is the key feature allowing the framework to be applied for estimating algorithmic features [25], like the algorithmic probability and algorithmic complexity. Note that, this parallelism we propose is different from the superposition of inputs as considered by standard QTM models [80], as the parallelism we propose allows both the input data as well as the transition function to be in a superposition. As an analogy, this can be thought of as the distinction between a quantum adder (which can add a superposition of two inputs), and a quantum calculator (which can apply all possible binary operations like add, subtract, multiply, divide, power, etc. on the two inputs). Each classical binary operation need not access the inputs from the tape in a coordinated fashion, e.g. multiplication can proceed from right to left while division can proceed from left to right. Once the quantum algorithm is executed, the output evolves to a superposition considering all possible programs (with the given states and symbols) on all possible inputs - commonly referred to as the Solomonoff universal prior probability distribution in algorithmic information theory.

Bounds on runtime

For any pragmatic (quantum) automata implementation, besides the memory, the runtime also needs to be restricted to a predetermined cycle count. This is imperative as it is not possible to estimate the halting time of a Turing machine in advance. Not all applications however allow a time restricted formulation. Thus, we are interested in these specific types of algorithms which can be aborted externally (without inspecting the current progress). The premature output should yield an acceptable approximate solution with a bounded error based on the computation time. These come under the purview of soft-computing and are ubiquitous in optimization, machine learning and evolutionary algorithms.

Here we consider runtime restriction in the context of a linear-bounded automata. There are 2 variables in a LBA specification, apart from the alphabet and states. These are the length of computation (time) and the length of the tape (space) resources. The latter is the dependent variable due to the region of influence (time cone). It is possible for a code to run for infinite time on a limited tape, but it is not possible to cover an infinite tape in a finite time. Thus, time-restricted automata further restrict the set of strings that are reachable by the computation process. Due to their restricted power, these sub-universal models are traditionally not preferred in theoretical computer science proofs where the typical assumption allows sufficient time and memory resource for the algorithm to complete its intended purpose. However, our assumptions on the space and time resources are motivated from the application perspectives of biological processes that we intend to model. Physical processes in nature can neither afford infinite time (cycles), not infinite space (memory), thus, we consider a computational model with realistic bounds on these variables.

There is no definitive metric to compare the power of runtime restricted automata models without invoking algorithmic information definitions like the speed prior or logical depth. Most landmark research [101] 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. The runtime of our model is thus application and resource driven, based on the convergence of the model for the specific phenomena, and the coherence limits of the quantum processor.

Additionally, we do not consider halting states specifically in our model. However, it is not too difficult to consider the subset of programs with halting states and are by default realized when a state loops on itself. Recent research [102] has shown 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 presume that it would also be correlated to LBA rather than to PDA or FSM. As discussed later, restricting the automata based on the number of cycles is an unavoidable technicality in the quantum version as we need to collapse the superposition.

For restricted TM, 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.

CORRESPONDING CLASSICAL MODEL

Before we present the quantum model of QPULBA, in this section, we define the corresponding classical model where neither the program nor the input can be in a superposition. Thus, it corresponds to a restricted runtime ULBA, where the program is accepted from outside the system (e.g. stored in a program memory) and the input tape is finite in length.

In our computation model, we will restrict a m states, n symbols, 1 dimension tape Turing machine 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, resulting in a LBA.

The tape is initialized to the blank character as this does not reduce the computational power of the TM. 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 memory size of CPU, 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 $c \leq (2t+1)$.

The generic model of a restricted ULBA is represented by this 10-tuple.

$$T = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F, t, d, c \rangle$$

- Q is a finite, non-empty set of states.
- Γ is a finite, non-empty set of symbols allowed on the tape, called the tape alphabet.
- $b \in \Gamma$ is the blank symbol.
- $\Sigma \subseteq \Gamma \setminus \{b\}$ is the set of input symbols, that is, the set of symbols allowed to appear in the initial tape contents.
- δ is a partial function called the transition function. It defines the next state, tape movement and write symbol based on the current state and the read symbol.
- $q_0 \in Q$ is the initial state.
- $F \subseteq Q$ is the set of final states or accepting states.
- t is the number of steps the machine is executed before a forced halt is imposed.
- d is the dimension of the tape. It also specifies if the tape is circular by a $\circ$ symbol for each dimension.
- c is the length of the tape in each dimension.

For our experimental implementation example in this paper, we chose $c = t$ and a circular tape. This helps us evaluate output diversity considering tape direction equivalence under left-right substitution. This shorter tape however comes as the cost of not able to map to computational path dependent application where left-right substitution is not always equivalent (e.g. robotic movement). Our computation model of $m = |Q|$ states, $n = |\Gamma|$ symbols, $d = 1^{\circ}$ dimension (circular) tape restricted Turing machine, can be formally represented as:

$$T = \langle \{Q_0, Q_1, \ldots, Q_{m-1}\}, \{s_0, s_1, \ldots, s_{n-1}\}, s_0, \{\}, \delta, Q_0, \{\}, c, 1^{\circ}, c \rangle$$

Note that Σ is empty, thus the tape is always initialized to the blank character $b = s_0$. The set of accepting states F is also empty to prevent the machine from halting before t steps. This includes machines that have defined halting states, by modifying the transition function to remain in the same state and write the same symbol that is read (in effect simulating a no-operation) once these states are reached.

Thus, the transition table exhaustively lists a transition for each combination of $(Q - F) = Q$ and Γ.

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{0, 1\}$$

where, 0 is left shift, 1 is right shift for the 1 dimensional tape. Each entry of the transition table requires $log_2(m) + log_2(n) + d$ bits and there are a total of $m * n$ entries. Thus, the number of bits required to specify a single transition function is:

$$q_{\delta} = (m * n) * (log_2(m) + log_2(n) + d)$$

The number of different machines (programs) that can be represented using q_δ bits is represented by:

$$P = 2^{q_\delta}$$

3.3.4. ENUMERATION OF EXAMPLE CASES

Here, some examples of this computation model are enumerated. These enumerations are intended to be executed in parallel thereby presenting the PULBA variant of the ULBA model. The cases are labeled as per the number of states, symbols and tape dimension (Case m-n-d). We start with the smallest natural number 1 and explore larger values of symbols and states (however, we will only consider 1 dimensional Turing tape).

For these experiments, the number of iterations is set equal to the size of the program, i.e. $t = q_\delta$. This allows us to compare the final tape and program to infer which programs can self-replicate, which motivates our research [25].

CASE 1-1-1

For this case, the number of states $m = 1$ with the state set $Q : \{Q_0\}$. The alphabet is $\Gamma : \{0\}$, thus, $n = 1$ (the unary alphabet). This gives the values $q_\delta = 1 * 1 * (0+0+1) = 1$ and $P = 2^1 = 2$ using the formula discussed before.

The machine is run for $t = q_\delta = 1$ iteration. The initial tape is of length $c = t = 1$. To distinguish the blank character from a written 0, we will use o. Thus, the initial tape is: o.

The outputs for each program (description number) are listed in Table 3.2. R_s/W_s refers to the read/write symbols and $M_{l/r}$ refers to the tape movement of left/right.

Table 3.2: Exhaustive enumeration of the programs of Case 1-1-1 circular tape ULBA for length 1 cycle 1.

P#	$Q_0 R_0$	Final tape
0	$Q_0 M_l W_0$	0
1	$Q_0 M_r W_0$	0

CASE 2-1-1

For this case, the number of states $m = 2$ with the state set $Q : \{Q_0, Q_1\}$. The alphabet is again $\Gamma : \{0\}$, with, $n = 1$ (the unary alphabet). This gives the values $q_\delta = 2 * 1 * (1+0+1) = 4$ and $P = 2^4 = 16$.

The machine is run for $t = q_\delta = 4$ iteration. The initial tape is of length $c = t = 4$. The initial tape is: $\underline{o}ooo$, where the underline denotes the initial position.

The outputs for each program (description number) are listed in Table 3.3. It is easy to see all m-1-d cases will result in tapes with the blank/zero characters. The 0000 string is the only string in this language.

CASE 1-2-1

For this case, the number of states $m = 1$ with the state set $Q : \{Q_0\}$. The alphabet is $\Gamma : \{0, 1\}$, with, $n = 2$ (the binary alphabet). This gives the values $q_\delta = 1*2*(1+1+0) = 4$ and $P = 2^4 = 16$. The machine is run for $t = q_\delta = 4$ iteration. The initial tape is of length $c = t = 4$. The initial tape is similar to the last case: $\underline{o}ooo$.

Table 3.3: Exhaustive enumeration of the programs of Case 2-1-1 circular tape ULBA for length 4 cycle 4.

P#	$Q_1 R_0$	$Q_0 R_0$	Final tape
00	$Q_0 M_l W_0$	$Q_0 M_l W_0$	0000
01	$Q_0 M_l W_0$	$Q_0 M_r W_0$	0000
02	$Q_0 M_l W_0$	$Q_1 M_l W_0$	0000
03	$Q_0 M_l W_0$	$Q_1 M_r W_0$	00oo
04	$Q_0 M_r W_0$	$Q_0 M_l W_0$	0000
05	$Q_0 M_r W_0$	$Q_0 M_r W_0$	0000
06	$Q_0 M_r W_0$	$Q_1 M_l W_0$	0oo0
07	$Q_0 M_r W_0$	$Q_1 M_r W_0$	0000
08	$Q_1 M_l W_0$	$Q_0 M_l W_0$	0000
09	$Q_1 M_l W_0$	$Q_0 M_r W_0$	0000
10	$Q_1 M_l W_0$	$Q_1 M_l W_0$	0000
11	$Q_1 M_l W_0$	$Q_1 M_r W_0$	00o0
12	$Q_1 M_r W_0$	$Q_0 M_l W_0$	0000
13	$Q_1 M_r W_0$	$Q_0 M_r W_0$	0000
14	$Q_1 M_r W_0$	$Q_1 M_l W_0$	00o0
15	$Q_1 M_r W_0$	$Q_1 M_r W_0$	0000

Table 3.4: Exhaustive enumeration of the programs of Case 1-2-1 circular tape ULBA for length 4 cycle 4.

P#	$Q_0 R_1$	$Q_0 R_0$	Final tape
00	$Q_0 M_l W_0$	$Q_0 M_l W_0$	0000
01	$Q_0 M_l W_0$	$Q_0 M_l W_1$	1111
02	$Q_0 M_l W_0$	$Q_0 M_r W_0$	0000
03	$Q_0 M_l W_0$	$Q_0 M_r W_1$	1111
04	$Q_0 M_l W_1$	$Q_0 M_l W_0$	0000
05	$Q_0 M_l W_1$	$Q_0 M_l W_1$	1111
06	$Q_0 M_l W_1$	$Q_0 M_r W_0$	0000
07	$Q_0 M_l W_1$	$Q_0 M_r W_1$	1111
08	$Q_0 M_r W_0$	$Q_0 M_l W_0$	0000
09	$Q_0 M_r W_0$	$Q_0 M_l W_1$	1111
10	$Q_0 M_r W_0$	$Q_0 M_r W_0$	0000
11	$Q_0 M_r W_0$	$Q_0 M_r W_1$	1111
12	$Q_0 M_r W_1$	$Q_0 M_l W_0$	0000
13	$Q_0 M_r W_1$	$Q_0 M_l W_1$	1111
14	$Q_0 M_r W_1$	$Q_0 M_r W_0$	0000
15	$Q_0 M_r W_1$	$Q_0 M_r W_1$	1111

The outputs for each program (description number) are listed in Table 3.4. If we run the experiment for more than 4 steps then we will see more variety in the output.

CASE 2-4-1

Let us consider the case where the number of states $m = 2$ (with the states set $Q : \{Q_0, Q_1\}$) with the alphabet $\Gamma : \{A, C, G, T\}$, with, $n = 4$, inspired by the DNA alphabet. This gives the values $q_\delta = 2 * 4 * (1 + 2 + 1) = 32$ and $P = 2^{32} = 4294967296$. It is clear that even for the simple case of the DNA alphabet, an exhaustive search by enumeration is not possible on a classical algorithm (and thereby, also on a quantum computer simulator running on classical hardware). This exponential growth in the number of cases shows the difficulty of classical enumeration of ULBA motivating quantum acceleration [58, 103] for bioinformatics applications.

CASE 2-2-1

This 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 values $q_\delta = 2 * 2 * (1 + 1 + 1) = 12$ and $P = 2^{12} = 4096$ using the formula discussed before.

The machine is run for $t = q_\delta = 12$ iteration. The initial tape of length $c = t = 12$ is: $\underline{0}00000000000$

The program (description number) is encoded as:

$$[QMW]^{Q_1R_1}[QMW]^{Q_1R_0}[QMW]^{Q_0R_1}[QMW]^{Q_0R_0}$$

There are too many cases to enumerate by hand, so a Python script (the classical kernel we intend to accelerate) is written that emulates our restricted model of the linear bounded automata for all 4096 cases. The program can be found at the following link: https://github.com/Advanced-Research-Centre/QuBio/blob/master/Project_01/classical/

The tape output (universal distribution) for all the 4096 machines is plotted in Figure 3.4 while the algorithmic probability is listed in Figure 3.5.

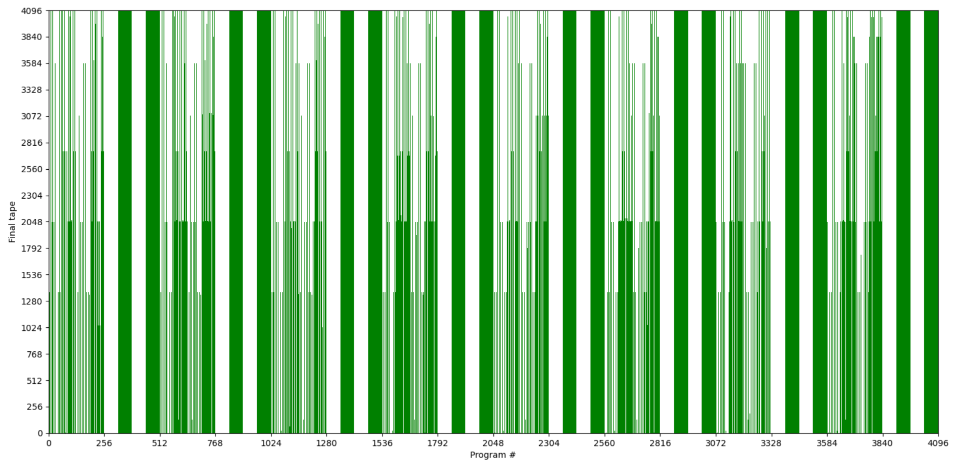


Figure 3.4: Tape output for all programs of Case 2-2-1 circular tape ULBA for length 12 cycle 12. This corresponds to the universal distribution.

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 3.5: Algorithmic probability (in red) for Case 2-2-1 (for non-zero probabilities).

3.3.5. Quantum parallel universal linear bounded automata

In this section, we present the detailed design of the quantum circuit to implement the ULBA computation model of the previous section. This is a mechanistic model of a QPULBA having the corresponding parts of a classical ULBA as qubits. The 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. As highlighted in Table 3.1, QPULBA (in yellow) captures the computing capabilities of 27 (in blue) out of 51 automata models (of type 3, 2, 1) that is realistically implementable on physical hardware. The circuit in Figure 3.6 requires some ancilla qubits which will be introduced later. The automata step needs to be repeated for the number of steps t we intend to execute the machine before measuring out the qubits.

Qubit complexity

The qubit complexities of the design elements are discussed here. The generic formula is derived before applying to the specific case of the 2-2-1 QPULBA.

- **Alphabet**: Alphabet set cardinality $n = |\Gamma|$ is the number of symbols in the alphabet. The number of bits/qubits required to represent a symbol, $q_\Gamma = \lceil log_2(n) \rceil$
- **Head position**: The current head position is represented either as binary or one-hot encoding. The one-hot encoding is more expensive in the number of qubits, but better in terms of gates. The number of bits/qubits required for one-hot encoding [80] is the same as the number of cells c. Since the simulation bottleneck is the number of qubits instead of the number of gates, we prefer the binary encoding. For binary encoding, $q_{head} = \lceil log_2(c) \rceil$
- **Read head**: The read head temporarily stores the content of the current head position, requiring bits/qubits, $q_{read} = q_\Gamma$
- **Write head**: The write head temporarily stores the content to be written to the

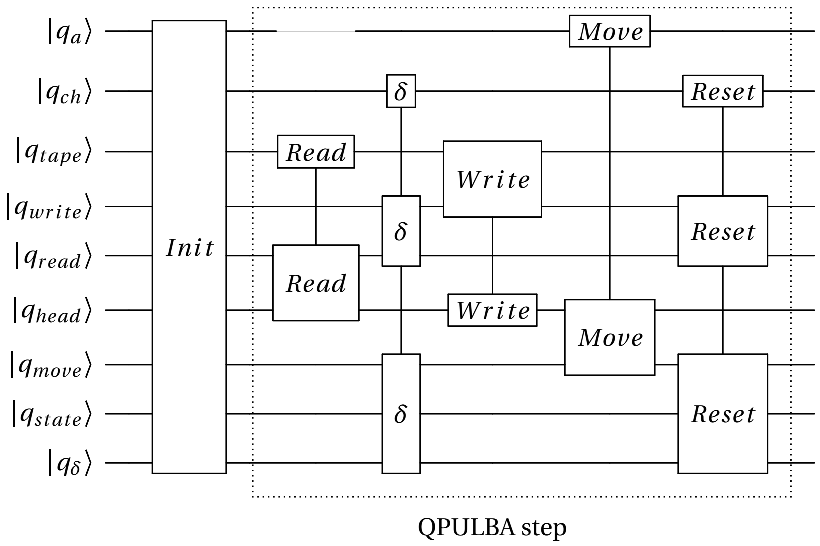


Figure 3.6: Blocks for the quantum circuit implementation of a QPULBA step.

current head position, requiring bits/qubits, $q_{write} = q_\Gamma$

- **Turing tape**: The number of bits/qubits required for the restricted tape size of c is, $q_{tape} = c * q_\Gamma$
- **Movement**: Specifying the movement of a d dimensional Turing tape requires, $q_{move} = d$
- **Current state**: The current state in binary encoding requires, $q_{state} = \lceil log_2(m) \rceil$. In a one-hot coded format, it would require m qubits.
- **Transition table**: The transition function is a unitary matrix that transforms the input and the current state to the output, next state and movement of the tape. Thus, for each combination of state and read character, we need to store the next state, write character and movement. The number of qubits required are, $q_\delta = (m * n) * (q_{state} + q_{write} + q_{move})$
- **Computation history**: Since the quantum circuit is reversible, the computation history for $(t - 1)$ steps needs to be stored in ancilla qubits. The computation history is specified by the state and read symbol for each step, requiring, $q_{ch} = (t - 1) * (q_{state} + q_{read})$. However, it is possible to trade-off space (qubits) with time (operations) by uncomputing.

Thus, the qubit complexity of the implementation (assuming q_a ancilla qubits) is:

$$\begin{aligned} q_{QPULBA} &= q_\delta + q_{state} + q_{move} + q_{head} + q_{read} + q_{write} + q_{tape} + q_{ch} + q_a \\ &= (m * n * (log(m) + log(n) + d)) + log(m) + d + log(c) \\ &+ log(n) + log(n) + (c * log(n)) + q_{ch} + q_a \end{aligned}$$

Considering the 2-2-1 QPULBA example, the values of $m = 2$, $n = 2$, $d = 1$, $c = 12$, $t = 12$ are substituted in the above equation (all logarithms are base-2 and rounded up to be nearest integer) to yield,

$$
\begin{aligned}
q^{221}_{QPULBA} &= (2*2*(log(2)+log(2)+1))+log(2)+1+log(12)+log(2)+log(2) \\
&\quad +(12*log(2))+(11*(log(2)+log(2)))+q_a \\
&= 12+1+1+4+1+1+12+22+q_a \\
&= 54+q_a
\end{aligned}
$$

Simulating in order of 50 qubits is near the quantum supremacy limits. However, the circuit is not always in full superposition thereby allowing smart simulation techniques in a quantum simulator and uncomputing away the computation history.

INITIALIZE

The initialization circuit depends on the target application for this framework. 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 3.7.

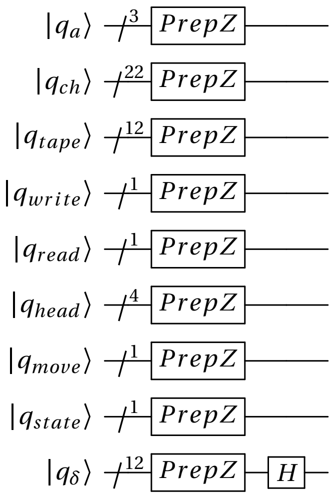

$|q_a\rangle$ 3 PrepZ
$|q_{ch}\rangle$ 22 PrepZ
$|q_{tape}\rangle$ 12 PrepZ
$|q_{write}\rangle$ 1 PrepZ
$|q_{read}\rangle$ 1 PrepZ
$|q_{head}\rangle$ 4 PrepZ
$|q_{move}\rangle$ 1 PrepZ
$|q_{state}\rangle$ 1 PrepZ
$|q_\delta\rangle$ 12 PrepZ H

Figure 3.7: Initialization quantum circuit for QPULBA 2-2-1.

QPULBA STEP

Each iteration of the QPULBA undergoes the following transforms:

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

This corresponds to one step of a classical UTM, with the distinction of the computation now evolving in a superposition of all possible classical automata. We will now discuss each of these QPULBA steps in detail.

READ TAPE

The quantum circuit implements a multiplexer with the tape as the input signals and the binary coded head position as the selector lines. The read head is the output. The read circuit for the QPULBA 2-2-1 is shown in Figure 3.8.

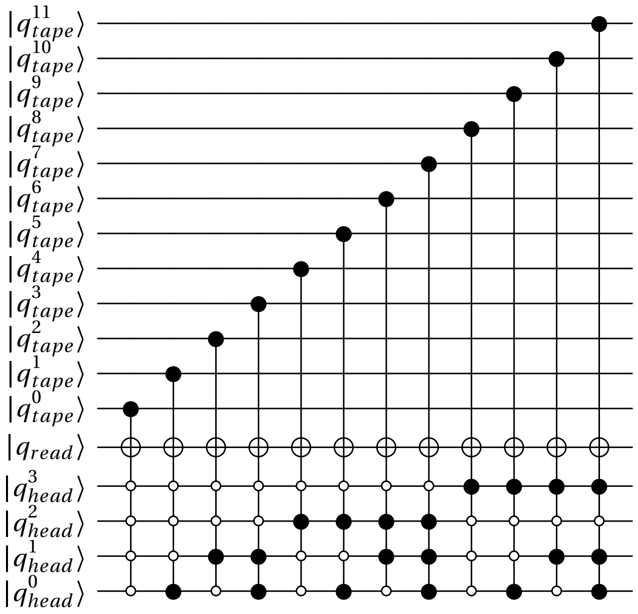
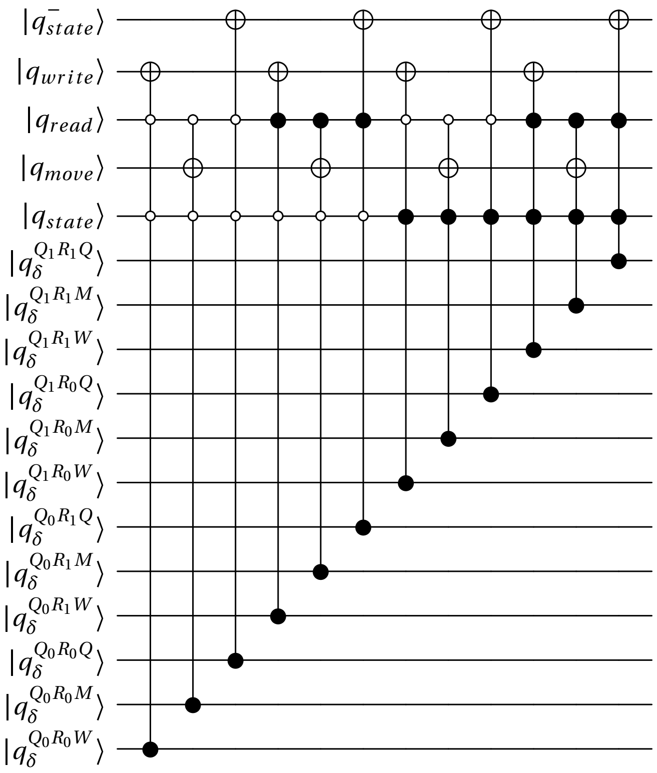


Figure 3.8: Read tape quantum circuit for QPULBA 2-2-1.

TRANSITION TABLE LOOKUP

The transition table encoding is: $[Q_t|R_\Gamma] \rightarrow [Q_{t+1}|M_{l/r}|W_\Gamma]$

Note that we use q^-_{state} instead of q^+_{state} though we are storing the next state in the qubit. This is corrected by the reset circuit. The transition function circuit for the QPULBA 2-2-1 is shown in Figure 3.9.



$|q^-_{state}\rangle$
$|q_{write}\rangle$
$|q_{read}\rangle$
$|q_{move}\rangle$
$|q_{state}\rangle$
$|q_\delta^{Q_1R_1Q}\rangle$
$|q_\delta^{Q_1R_1M}\rangle$
$|q_\delta^{Q_1R_1W}\rangle$
$|q_\delta^{Q_1R_0Q}\rangle$
$|q_\delta^{Q_1R_0M}\rangle$
$|q_\delta^{Q_1R_0W}\rangle$
$|q_\delta^{Q_0R_1Q}\rangle$
$|q_\delta^{Q_0R_1M}\rangle$
$|q_\delta^{Q_0R_1W}\rangle$
$|q_\delta^{Q_0R_0Q}\rangle$
$|q_\delta^{Q_0R_0M}\rangle$
$|q_\delta^{Q_0R_0W}\rangle$

Figure 3.9: Transition function quantum circuit for QPULBA 2-2-1.

WRITE TAPE

The quantum circuit implements a de-multiplexer with the tape as the output signals and the head position as the selector lines. The write head is the input.

The write circuit for the QPULBA 2-2-1 is shown in Figure 3.10. The qubit elements not involved are not shown.

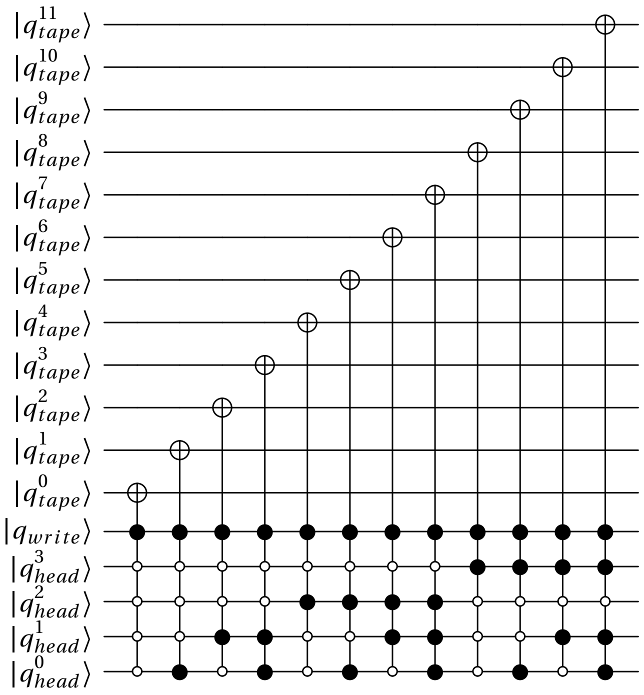
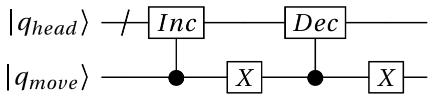


Figure 3.10: Write tape quantum circuit for QPULBA 2-2-1.

MOVE

There are many choices for implementing the move, e.g. a looped tape (overflow/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/decremented using the move qubit as control.



The increment/decrement circuit is a special case of the quantum full adder [104] 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 $(11+1)$ mod $12 = 0$, 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
cnot A0,S0
cnot B0,S0
```

```
.carry
toffoli A0,B0,C1
cnot A0,B0
toffoli C0,B0,C1
```

```
.reverse_carry
toffoli C0,B0,C1
cnot A0,B0
toffoli A0,B0,C1
```

Increment

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 3.11 shows the 4-bit modulo-12 quantum increment circuit using the quantum adder blocks.

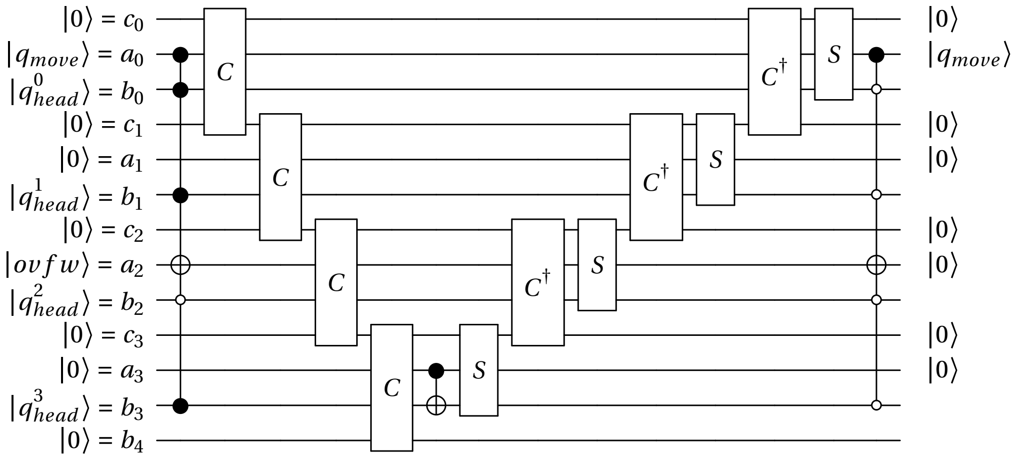


Figure 3.11: Modulo-12 quantum adder for implementing move tape head for QPULBA 2-2-1.

The circuit can be simplified considering some of the control qubits are always in the 0 state, so those CNOT/Toffoli gates can be ignored. The optimized increment circuit for the QPULBA 2-2-1 is shown in Figure 3.12.

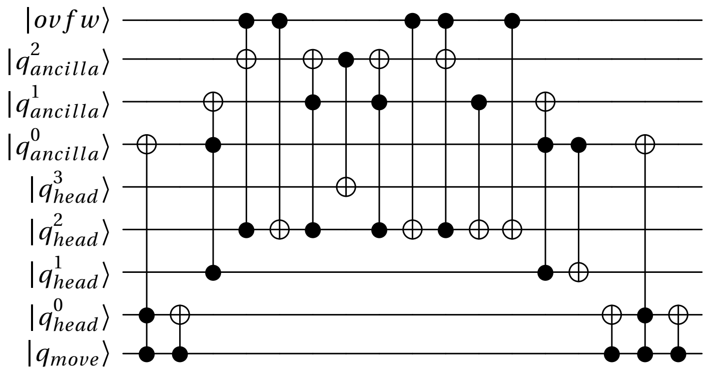


Figure 3.12: Modulo-12 increment quantum circuit for QPULBA 2-2-1.

Decrement

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 3.13 shows the 4-bit modulo-12 quantum decrement circuit using the reverse order of blocks as the standard quantum adder.

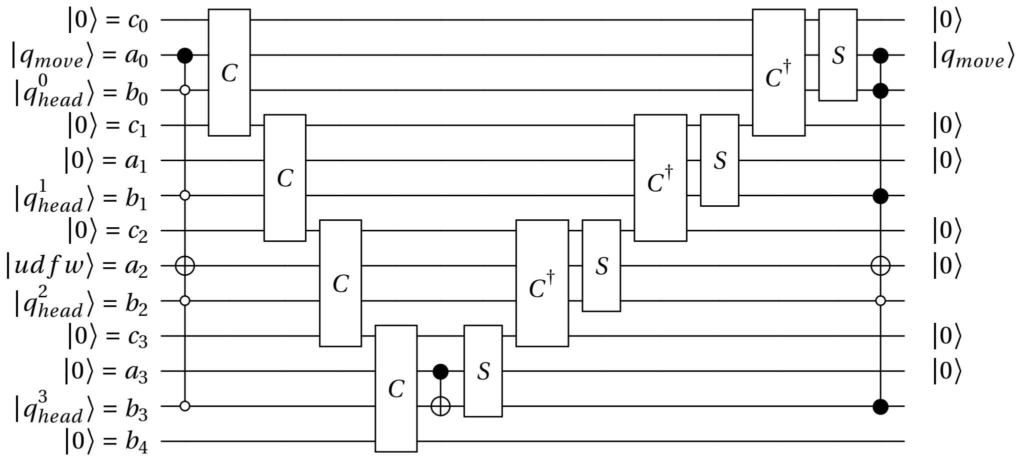


Figure 3.13: Modulo-12 quantum subtractor for implementing move tape head for QPULBA 2-2-1.

The circuit can be simplified considering some of the control qubits are always in the 0 state, so those CNOT/Toffoli gates can be ignored. The optimized increment circuit for the QPULBA 2-2-1 is shown in Figure 3.14.

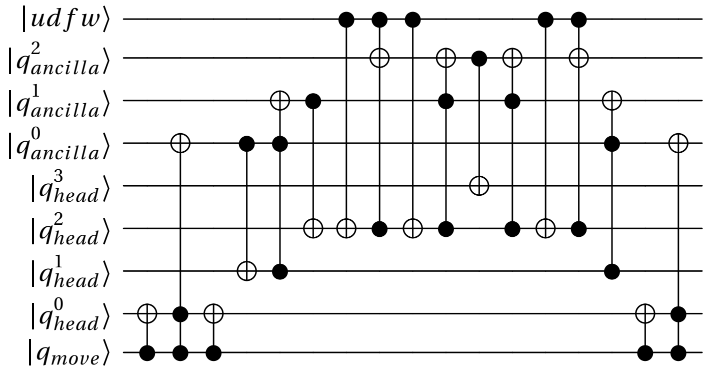


Figure 3.14: Modulo-12 decrement quantum circuit for QPULBA 2-2-1.

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, e.g. AND, extra qubits are needed to construct the reversible quantum circuit that preserves the unitary property. The general strategy to compile a classical function to quantum logic is shown in Figure 3.15.

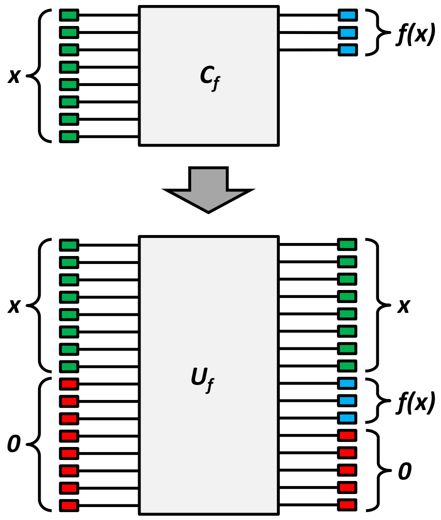


Figure 3.15: Reversible circuit compilation strategy.

Other research has focused on the quantum circuit generation of the constrained subset of reversible automata. 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.

3.3.6. IMPLEMENTATION AND SIMULATION RESULTS

In this section, we present the circuit implementation of QPULBA. This was implemented on 2 different programming platforms, OpenQL [105] developed at the Delft University of Technology and IBM's Qiskit [106]. Our copy-left AGPLv3 licensed implementation can be found on: https://github.com/Advanced-Research-Centre/QPULBA

We implemented 2 cases of QPULBA, with 1 and 2 states: the full circuit and cycle simulation of QPULBA 1-2-1, and a limited simulation of the units for QPULBA 2-2-1, as presented below.

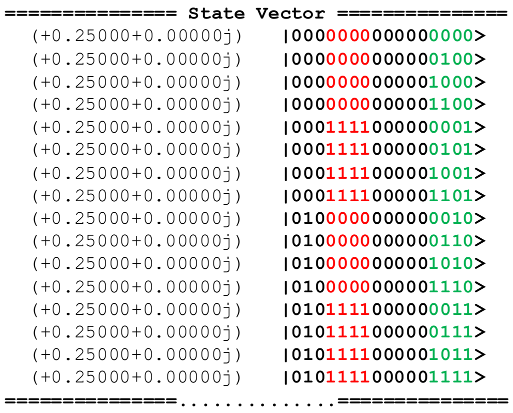
QPULBA 1-2-1

Our implementation 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. Note that there is no need to store the state anymore thereby reducing the qubit complexity greatly.

```
Number of 1-state 2-symbol 1-dimension QPULBA: 16

FSM     : [0, 1, 2, 3]
STATE   : []
MOVE    : [4]
HEAD    : [5, 6]
READ    : [7]
WRITE   : [8]
TAPE    : [9, 10, 11, 12]
ANCILLA : [13, 14, 15]
```

The full circuit was simulated for 4 cycles. The final state vector obtained after 4 cycles is shown in Figure 3.16. The FSM qubits encoding the description/program number (in green) and the output on the tape (in red) bit strings match with the classical enumeration in Table 3.4. 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.



```
=============== State Vector ===============
   (+0.25000+0.00000j)    |0000000000000000>
   (+0.25000+0.00000j)    |0000000000000100>
   (+0.25000+0.00000j)    |0000000000001000>
   (+0.25000+0.00000j)    |0000000000001100>
   (+0.25000+0.00000j)    |0001111000000001>
   (+0.25000+0.00000j)    |0001111000000101>
   (+0.25000+0.00000j)    |0001111000001001>
   (+0.25000+0.00000j)    |0001111000001101>
   (+0.25000+0.00000j)    |0100000000000010>
   (+0.25000+0.00000j)    |0100000000000110>
   (+0.25000+0.00000j)    |0100000000001010>
   (+0.25000+0.00000j)    |0100000000001110>
   (+0.25000+0.00000j)    |0101111000000011>
   (+0.25000+0.00000j)    |0101111000000111>
   (+0.25000+0.00000j)    |0101111000001011>
   (+0.25000+0.00000j)    |0101111000001111>
===============.............===============
```

Figure 3.16: Test result for the QPULBA 1-2-1 showing the FSM description/program number (green) and output tape (red).

QPULBA 2-2-1

The entire circuit for the 2-state 2-symbol case can be compiled from the parts described before. This was implemented in a scalable manner on OpenQL and Qiskit.

Full circuit compilation

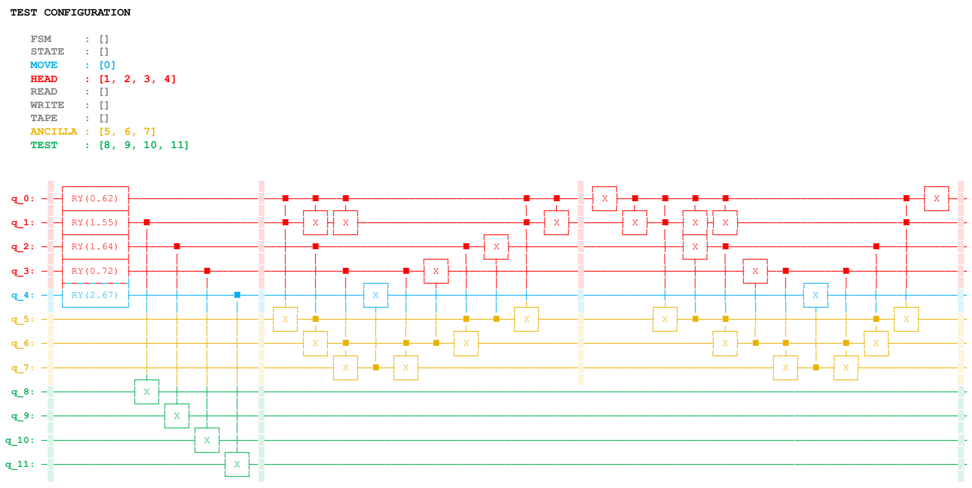
The qubit allocation for the full circuit, considering 1 cycle is:

```
Number of 2-state 2-symbol 1-dimension QPULBA: 4096

FSM      : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
STATE    : [12, 13]
MOVE     : [14]
HEAD     : [15, 16, 17, 18]
READ     : [19]
WRITE    : [20]
TAPE     : [21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32]
ANCILLA  : [33, 34, 35]
```

For each further cycles, we need 2 qubits to store the computation history.

The exact gate complexity depends on the considered primitive. We use Hadamard, Pauli-X, CNOT, Toffoli and SWAP as our gate set. Multi-qubit controlled-NOT gates are decomposed using the borrowed-ancilla strategy outlined in [52]. One cycle of the QPULBA uses 627 gates: 476 Toffoli, 126 Pauli-X, 12 CNOT, 12 Hadamard and 1 SWAP gate. The Qiskit circuit drawing and generated OpenQASM can be found on the repository.



```
TEST CONFIGURATION

    FSM      : []
    STATE    : []
    MOVE     : [0]
    HEAD     : [1, 2, 3, 4]
    READ     : []
    WRITE    : []
    TAPE     : []
    ANCILLA  : [5, 6, 7]
    TEST     : [8, 9, 10, 11]
```

Figure 3.17: Test circuit for the QPULBA 2-2-1 move block.

Unit tests

While we were able to compile the full circuit, the large number of qubits limits classically simulating the circuit on our available hardware. The exponential simulation complexity of quantum algorithms on classical hardware in terms of memory resource is indeed the driving factor for research on physical implementation of quantum accelerators. To complement our design, we developed unit tests for each part of the mechanistic perspective.

```
============ State Vector ============
  (+0.03251+0.00000j)    |000000000011>
  (+0.10148+0.00000j)    |000000011110>
  (+0.09940+0.00000j)    |000100000000>
  (+0.03184+0.00000j)    |000100000101>
  (+0.10876+0.00000j)    |001000000010>
  (+0.03484+0.00000j)    |001000000111>
  (+0.10652+0.00000j)    |001100000100>
  (+0.03412+0.00000j)    |001100001001>
  (+0.03820+0.00000j)    |010000000110>
  (+0.01224+0.00000j)    |010000001011>
  (+0.03741+0.00000j)    |010100001000>
  (+0.01198+0.00000j)    |010100001101>
  (+0.04094+0.00000j)    |011000001010>
  (+0.01311+0.00000j)    |011000001111>
  (+0.04010+0.00000j)    |011100001100>
  (+0.01284+0.00000j)    |011100010001>
  (+0.42239+0.00000j)    |100000001110>
  (+0.13530+0.00000j)    |100000010011>
  (+0.41369+0.00000j)    |100100010000>
  (+0.13252+0.00000j)    |100100010101>
  (+0.45268+0.00000j)    |101000010010>
  (+0.14500+0.00000j)    |101000010111>
  (+0.44336+0.00000j)    |101100010100>
  (+0.14202+0.00000j)    |101100011001>
  (+0.15899+0.00000j)    |110000010110>
  (+0.05093+0.00000j)    |110000011011>
  (+0.15571+0.00000j)    |110100011000>
  (+0.04988+0.00000j)    |110100011101>
  (+0.17039+0.00000j)    |111000011010>
  (+0.05458+0.00000j)    |111000011111>
  (+0.05346+0.00000j)    |111100000001>
  (+0.16688+0.00000j)    |111100011100>
============.............============
```

Figure 3.18: Test result for the QPULBA 2-2-1 move block showing the move (blue), head (green), ancilla (yellow) and test (green) qubits.

We successfully tested each of the 6 parts of the QPULBA, i.e. initialize, read, FSM, write, move and reset. The unit test simulations are tractable as each part concerns only a subset of the qubits. The inputs are put into an unequal superposition of values using random rotations about the Y-axis (that maintains the amplitude in the real domain). This allows us to individually inspect each basis state changes in contrast to an equal superposition using Hadamard gates. For quantum acceleration of classical algorithms, the ZX-plane of the Bloch sphere is enough to take advantage of the destructive interference of quantum superpositions over classical probabilistic computing, thereby reducing the cost of classical simulation using complex representations of the amplitude.

To track the output changes, each qubit of the target register is entangled with a test register using CNOT gates. Thus, the old value and the new value of the basis state's bit string can be inspected similar to an associative memory.

Here, as an example, we show the unit test circuit for the move step, which requires 8 qubits for the circuit and 4 addition qubits for testing. The circuit is shown in Figure 3.17. The initial part before the first barrier is the test configuration and the second part is of the move circuit.

For the move circuit, the binary string of the qubits associated with the head state (in red) increments by 1 if the move qubit (in blue) is 1 and decrements by 1 otherwise. This is verified by the internal state vector output of the simulation as shown in Figure 3.18.

3.4. ALGORITHMIC INFORMATION

Algorithmic information theory (AIT) [107] 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 [108], Andrey Kolmogorov [109], and Gregory Chaitin [110] looked at information, probability, and statistics through the algorithmic lens. The theory has now become a central part of theoretical computer science [111].

While the applicability of mathematical objects defined in the context of AIT are ubiquitous, most 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 remains very limited. The definitions and applications of some of these mathematical objects in AIT that are used in this research, are explained here. While for some of the definitions herein, a generic non-universal TM is sufficient, based on our use case of AIT, we will consider the forms based on UTM unless otherwise required.

- **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:

  $$AC_U(s) = \min:\{l(p), U(p) \rightarrow s\}$$

  where U is a UTM, p is a program, $l(p)$ is the length of the program, s is a string, and $U(p) \rightarrow s$ denotes the fact that p executed on U 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 UTM 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 (e.g., MetaGolfScript). However, the invariance theorem guarantees that, there exists an additive constant $c_{U1\rightarrow U2}$ independent of s, such that

  $$AC_{U2} \leq AC_{U1} + c_{U1\rightarrow U2}$$

This constant is the compiler length for translating any arbitrary program p_{U1} for $U1$ to p_{U2} for $U2$.

- **Algorithmic probability**: Algorithmic probability (AP), also called Solomonoff's probability, is the chance that a randomly selected program will output s when executed on U. The probability of choosing such a program is inversely proportional to the length of the program.

$$AP_U(s) = \sum_{p:U(p)\to s} 2^{-l(p)}$$

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 [112] to AC via the following law:

$$AC_U(s) \leq -log_2(AP_U(s)) + c$$

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 p on a TM T 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:

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

The coding theorem method (CTM) [113] approximates the AC as:

$$CTM(n,m)(s) = -log_2 D(n,m)(s)$$

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) [77] approximates the CTM value for an arbitrarily large object by decomposing it into smaller slices of appropriate sizes for which CTM values are known and aggregated back to a global estimate. The algorithmic complexities of these small slices are precomputed and made into a look-up table using CTM. The BDM aggregates this as:

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

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 in the method of dividing the string into blocks become negligible [114] in the limit, e.g., to take a sliding window at the boundary or an overlapping block.

- **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.

- **Universal search**: Although the UD is uncomputable, universal search (US), also called Levin search [115], converges to the universal distribution when executed for longer periods of time. It solves inversion problems by interleaving the execution of all possible programs on a universal Turing machine, sharing computation time equally among them until one of the executed programs manages to solve the given inversion problem.

  US has inspired various artificial general intelligence (AGI) approaches that built upon this to calculate the expected value of an action. The more computing power that is available, the closer their predictions are to the predictions of inductive inference. Jürgen Schmidhuber and Marcus Hutter developed many AGI algorithms like Adaptive Levin Search, Probabilistic Incremental Program Evolution, Self-Modifying Probabilistic Learning Algorithms, Hutter Search, Optimal Ordered Problem Solver, AIXI-tl, and Gödel Machine. These methods will be explored as an extension of this research.

- **Levin complexity**: Levin complexity (LC) is a computable (though often intractable), time-bounded version of AC which penalizes execution time t.

  $$LC_U(s,t) = \min:\{l(p) + log(t), U(p) \rightarrow s\}$$

  The standard invariance theorem also holds for this.

- **Logical depth**: Bennett's logical depth (LD) [116] is the number of time steps t to run the shortest program p_{AC} to output s in a specific UTM U. It captures the general notion of physical complexity, which implies that neither random nor simple objects are sophisticated (complex).

  $$LD_U(s) = \min:\{t(p_{AC}) : U(p_{AC}) \rightarrow s\}$$

  The choice of the TM has a multiplicative factor for LD is more crucial.

  In general, it is not a semi-computable measure (approximation can be bounded from a direction) but can be approximated based on the decompression time of compression algorithms. For a given significance level d, LD can be calculated as:

  $$LD_U(s,d) = \min:\{(t(p) : U(p_{AC}) \rightarrow s) \wedge (l(p) - l(p_{AC}) \leq d)\}$$

- **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 [117] the measure speed prior (SP), derived from the fastest way of computing data.

$$SP_U(s) = \sum_{z=1}^{\infty} 2^{-z} \sum_{p:U(p,z)\to s} 2^{-l(p)}$$

  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 the 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.

$$\Omega_T = \sum_p 2^{-|p|}$$

While many of the algorithmic objects 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 [77]. These approximations can further be calculated by time-bounding the TMs for a maximum number of cycles. Though time-bounded algorithmic information was 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 mathematical objects in AIT 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's) 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 is 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 to make 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, very few results for algorithmic information are developed. Recent research [118] on reinforcement learning shares a similar motivation; however, does not provide a detailed gate (time) and qubit (space) level analysis or implementation.

3.5. ALGORITHMIC PROPERTIES USING QUANTUM CIRCUITS

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 $c \leq (2t+1)$.

The detailed design of the quantum circuit to implement the computation model is presented in § 3.3. 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 [80],
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 [80]
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. a complete and scalable circuit description of the full construction in two popular quantum programming languages are provided with 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 for 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 3.19. The yin-yang symbol and its rotated forms denote qubits in superposition states.

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

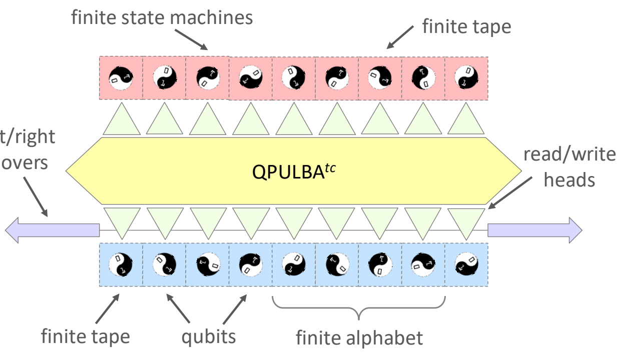


Figure 3.19: QPULBA^{tc}: Quantum Parallel Universal Linear Bounded Automata restricted by t cycles and tape length c.

on the Qiskit quantum programming language can be found at https://github.com/Advanced-Research-Centre/QPULBA. The blocks of the quantum circuit for a single QPULBA step is shown in Figure 3.6. The blocks need 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 this research, we focus on using the QPULBA quantum circuit for various applications.

3.5.1. 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 3.20. In this section, we will present the method to use U to estimate algorithmic information.

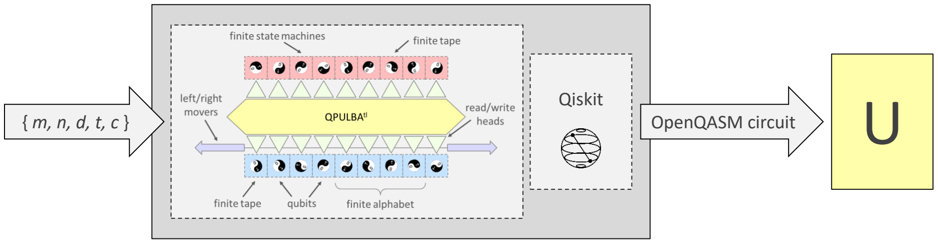


Figure 3.20: QPULBA circuit implementation, as detailed in [119].

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 information, 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 [120] 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 allow 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 an 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 problems for demonstrating speedup using quantum search.

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 3.4, 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.

FINDING OUTPUT WITH HIGHEST ALGORITHMIC PROBABILITY

The mapping between programs and output can be represented as a bipartite directed graph with a many-to-one relation. The output with the 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.

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 information 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 [49]. 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 the following sections.

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.

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.

3.5.2. 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.

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 3.21, 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 [121] or an instance of an evolving neural network [122].

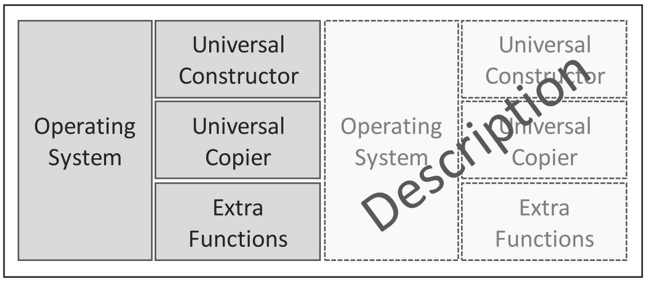


Figure 3.21: John von Neumann's system of self-replicating automata.

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.

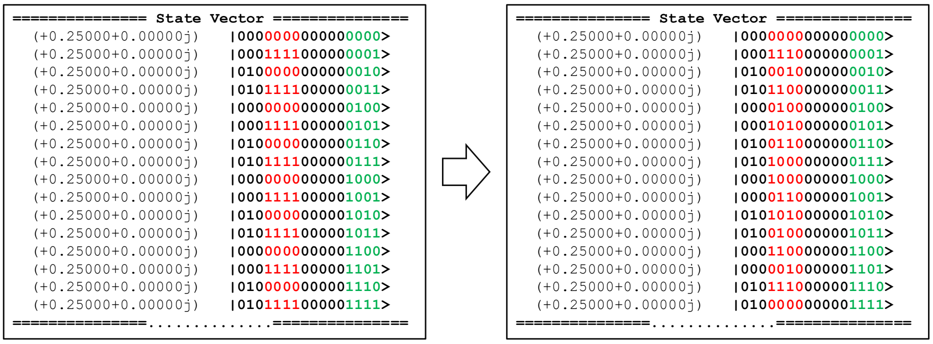
The idea of using the fixed-point, called the Y-combinator in lambda calculus $\lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$ to describe the genetic code [123] is pioneered by Gregory Chaitin [78] as the field meta-biology. This is in line with the constructor theory [124] approach for understanding 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 [125]. In our future research [126], we intend to explore the significance of these results for artificial life applications [127] in synthetic biology. The space and probability of constructors [128] can inform the subset of DNA encoding for in vitro experimentation and understanding of causal mechanisms in cells [79, 129].

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. The code can be found at https://github.com/Advanced-Research-Centre/QPULBA. The final state vector obtained after four cycles is shown on the left side of Figure 3.22. 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 3.22.



```
=============== State Vector ===============
  (+0.25000+0.00000j)    |000000000000000000>
  (+0.25000+0.00000j)    |0001111000000001>
  (+0.25000+0.00000j)    |0100000000000010>
  (+0.25000+0.00000j)    |0101111000000011>
  (+0.25000+0.00000j)    |0000000000000100>
  (+0.25000+0.00000j)    |0001111000000101>
  (+0.25000+0.00000j)    |0100000000000110>
  (+0.25000+0.00000j)    |0101111000000111>
  (+0.25000+0.00000j)    |0000000000001000>
  (+0.25000+0.00000j)    |0001111000001001>
  (+0.25000+0.00000j)    |0100000000001010>
  (+0.25000+0.00000j)    |0101111000001011>
  (+0.25000+0.00000j)    |0000000000001100>
  (+0.25000+0.00000j)    |0001111000001101>
  (+0.25000+0.00000j)    |0100000000001110>
  (+0.25000+0.00000j)    |0101111000001111>
===============.............===============
```

```
=============== State Vector ===============
  (+0.25000+0.00000j)    |0000000000000000>
  (+0.25000+0.00000j)    |0001110000000001>
  (+0.25000+0.00000j)    |0100010000000010>
  (+0.25000+0.00000j)    |0101100000000011>
  (+0.25000+0.00000j)    |0000100000000100>
  (+0.25000+0.00000j)    |0001010000000101>
  (+0.25000+0.00000j)    |0100110000000110>
  (+0.25000+0.00000j)    |0101000000000111>
  (+0.25000+0.00000j)    |0001000000001000>
  (+0.25000+0.00000j)    |0000110000001001>
  (+0.25000+0.00000j)    |0101010000001010>
  (+0.25000+0.00000j)    |0100100000001011>
  (+0.25000+0.00000j)    |0001100000001100>
  (+0.25000+0.00000j)    |0000010000001101>
  (+0.25000+0.00000j)    |0101110000001110>
  (+0.25000+0.00000j)    |0100000000001111>
===============.............===============
```

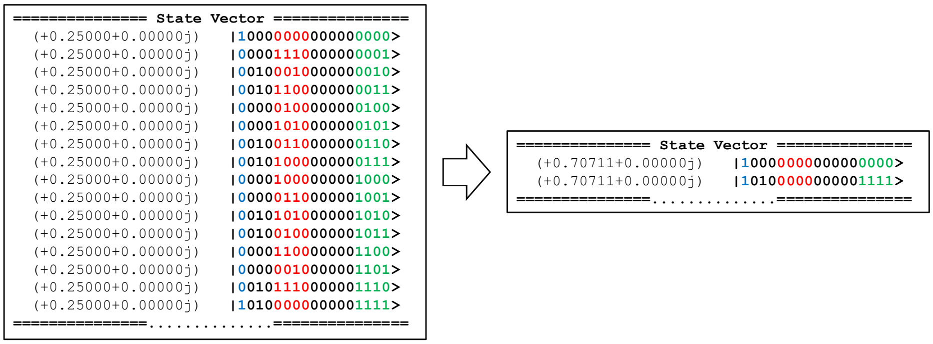
Figure 3.22: QPULBA 1-2-1 showing the FSM description/program number (green) and output tape (red): on left the universal distribution and on right the Hamming distance between tape and FSM.

The 0000 tape state is then marked on the MSQ (in blue), as shown on the left size of Figure 3.23. On sampling the 1 state of this qubit evolves the quantum state to an equal 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 $O(t*(p/p_q))$.

ESTIMATION OF ALGORITHMIC COMPLEXITY

In [77], 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 [130], psychometrics [131], cellular automata [132, 133], graph theory, economic



```
=============== State Vector ===============
  (+0.25000+0.00000j)    |10000000000000000>
  (+0.25000+0.00000j)    |00001110000000001>
  (+0.25000+0.00000j)    |00100010000000010>
  (+0.25000+0.00000j)    |00101100000000011>
  (+0.25000+0.00000j)    |00000100000000100>
  (+0.25000+0.00000j)    |00001010000000101>
  (+0.25000+0.00000j)    |00100110000000110>
  (+0.25000+0.00000j)    |00101000000000111>
  (+0.25000+0.00000j)    |00001000000001000>
  (+0.25000+0.00000j)    |00000110000001001>
  (+0.25000+0.00000j)    |00101010000001010>
  (+0.25000+0.00000j)    |00100100000001011>
  (+0.25000+0.00000j)    |00001100000001100>
  (+0.25000+0.00000j)    |00000010000001101>
  (+0.25000+0.00000j)    |00101110000001110>
  (+0.25000+0.00000j)    |10100000000001111>
===============..............===============
```

```
=============== State Vector ===============
  (+0.70711+0.00000j)    |10000000000000000>
  (+0.70711+0.00000j)    |10100000000001111>
===============..............===============
```

Figure 3.23: Sampled distribution of quines.

time series [134, 135], and complex networks [131]. 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 previous sections, this is an absolute measure of the information content of a string. A seminal result in estimating AC [136] via Experimental Algorithmic Information Theory (EAIT)-based approach considers the property that many of the algorithmic mathematical objects 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 defining 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 discussed. 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 information 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 information 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) [137] 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 ≤ 12 over ≈ 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 [138] 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 [77].

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 [139]. Toda's theorem states the entire polynomial hierarchy PH is contained in P^{PP}. Since $P^{PP} = P^{\#P}$ [85] 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 [140] 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 [141] 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 [142] 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 [143] 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 [118] 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 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

```
1 Given string $s$;
2 for $i < t$ do
3     Run QPULBA step;
4 end
5 $num_s := QCount(tape = s)$;
6 for $i < t$ do
7     Run QPULBA step;
8 end
9 $num_h := QCount(state = halt)$;
  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 [38] 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 [144–146]. Implementing these on the Qiskit framework and integrating with the CTM algorithm will be explored in our future work.

3.5.3. APPLICATION FRAMEWORK

The model of computation discussed in this research can estimate time-bound algorithmic information. Here, we present a framework for empirical experimentation on a quantum accelerator. This is shown in Figure 3.24. The output of the unitary U implementing the time-bound QPULBA model can be conditioned based on the required application. The three advantageous cases: 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's search, or near-term variational quantum optimization approaches using parametric circuits such as QAOA [67, 147, 148]. 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 information such as Kolmogorov complexity is widespread [111], 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 meta-mathematics by Gregory Chaitin). It involves enumerating and running programs to understand its statistical mechanics and is similar to Stephen Wolfram's metamathematics approach [149].

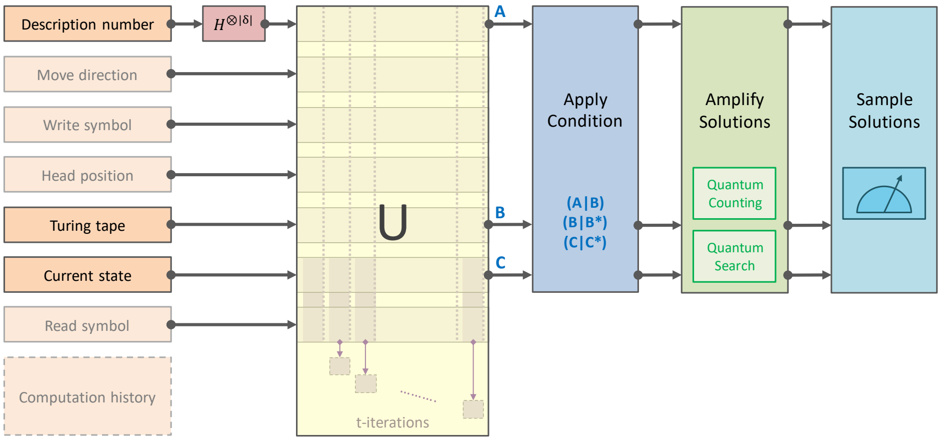


Figure 3.24: Application development framework for quantum experimental algorithmic information theory.

META-BIOLOGY AND ARTIFICIAL LIFE

Quantum EAIT can be used to accelerate meta-biology experiments in open-ended evolution [150] as busy beaver functions originally proposed in [78]. Meta-biology, as introduced by Gregory Chaitin, provides DNA linguistics [151–153] 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 [154] was pioneered by multi-disciplinary research in Hector Zenil's group, achieving impressive results [130] using CTM and BDM. Our research will complement most of the work [155] 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 [156] as part of this research.

In our past research, we developed quantum algorithms for accelerating DNA sequence reconstruction using both alignment [103]- 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.

Phylogenetic tree analysis using EAIT

Recently, AIT techniques were successfully applied [157] 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 additional information concerning these infectious agents and understand the dynamics of viral diseases.

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 [158] 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 [159] 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.

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 [160] 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.

3.6. Conclusions and outlook

The state of current quantum hardware is at a crucial junction of quantum supremacy, where a problem that has mathematical guarantee of quantum speedup can be demonstrated on QPU platforms. However, there exists a huge gap between the resource requirements of quantum supremacy and quantum advantage, where a societal/industrial use case can be implemented for a problem size that is intractable in classical infrastructures like supercomputers.

This gap has driven the community towards parametric quantum computing (PQC), which require much less qubits and are more noise tolerant. However, it comes with its own set of caveats.

- Variational quantum circuits are similar to approaches like genetic algorithms or neural networks, where the performance of a proof-of-concept cannot be extrapolated when scaled up for a different size or training set. There is also not much quantum logic design involved from an algorithm perspective, beyond formulating the problem as an optimization instance, some design choices (like ansatz and the classical optimizer), and related hyper-parameter tuning.
- Problem driven ansatz are often not error tolerant, leading to problems like the optimization process getting stuck at plateaus in the optimization landscape. Also, the results are heavily dependent on the characteristics and power of the classical optimizer. Both ansatz design and PQC optimizer design are active yet underdeveloped research fields for off-the-shelf use in quantum application development. Digressing into these is beyond the scope of this research.
- The goal of NISQ era is to extract as much computing power as possible by fine tuning the algorithm to the specific hardware. Since the hardware-driven ansatz and parameter optimization process depend on the specific QPU, it is counter-productive to develop a perfect qubit formulation on a simulator (as is the scope of this research) without demonstrating it on real hardware. Besides, even for PQC, larger and better QPU are required for demonstrating quantum advantage in genoinformatics problems.
- Being heuristic, NISQ approaches are not mathematically provable, rather empirically demonstrated. While these approaches have wide applications, there has also been some negative results (like poor performance for hard instances of SAT). The complexity of de novo assembly lie (with respect to the clause density of its TSP formulation of the overlap graph traversal), cannot be inferred without considering specific read size and real data sets. These make it a difficult choice to pick the PQC direction without experimental backing.

The two core algorithms explored in the previous chapter, QiBAM and QAOA promise at best a polynomial speedup for search (for reference-read alignment) and traveling salesman problem (for read assembly). The brute-force search approach often is the only approach in many applications where this Grover's search type speedup can be provably achieved on quantum and has justifiable benefits in a production scale environment. However, many industrial approaches (e.g. BWA-MEM, DBG) employ heuristics to drastically reduce the time complexity while still achieving acceptable approximations. In this chapter, a stronger guarantee is explored where the search database in unstructured, thus preventing heuristic or future classical algorithms from depleting the quantum advantage. Most problems in algorithmic information theory require enumerating the run behavior of Turing machines. These cannot be accelerated via algorithmic design on quantum or classical as any global behavior is uncomputable (due to Rice's theorem [161]). Quantum computing can however offer advantage by enumerating an exponential number of superposition of Turing machine, if the interest is in some statistical tendency of the final distribution of runtime behavior.

There is currently a thrust in research towards causal modeling instead of data-driven approaches. The applications include genomics and artificial intelligence among many others. The best theoretical method for causal modeling and inference is based on Solomonoff algorithmic probability and Bayesian probability respectively. Being uncomputable, these mathematical objects are currently approximated using classical supercomputers. These approximation algorithms, like the coding theorem method (CTM) and block decomposition method (BDM) are the classical kernels that motivated the quantum logic formulation of this chapter.

The mechanistic model of computation as exhibited by a Turing machine defines an algorithm as an initial input to final output transformation on the tape memory by a program defined as a finite state machine. The set of transformations a computation model can undergo and the resulting space of outputs is central to understanding the causal structure of a physical phenomena for scientific modeling and hypothesis testing. While it has many applications, except for the trivial cases, this remain intractable on classical computers. This is because the space of all possible transformations grows exponentially with the number of states and symbols of the automata. In this research, we explore the distinctive advantages for classical automata simulation offered by the alternate paradigm of quantum computation. We complement the recently proposed [80] circuit design of a quantum Turing machine from a mechanistic perspective with realistic assumptions on runtime and qubit resources. The proposed design presented in § 3.3 follows the computation model of a quantum parallel universal linear bounded automata. The exact scalable circuit is designed using standard quantum gates required to simulate a superposition of programs of this automata, thereby obtaining the distribution of their evolution after a predetermined number of cycles. We present our results of the implementation of two cases of the automata on Qiskit. We simulated and verified 4 cycles of the 1-state 2-symbol quantum parallel universal linear bounded automata with 16 qubits and compiled and unit tested the 2-state variant that requires 36 qubits.

Thereafter, § 3.5 presents a framework for empirically evaluating algorithmic information 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 mathematical objects 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. 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.

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. This algorithm can be readily ported on a quantum accelerator stack [6] with any sufficiently advanced gate-model quantum computing hardware, in terms of qubits, connection topology and error rates. 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.

4 Universal reinforcement learning in quantum environments

The exploration in § 3 revealed close links between the theoretical domains of quantum computation and algorithmic information. Specifically, this can be explored to find the best algorithm for a particular purpose. Quantum computing guarantees the most general (and fastest) type of computing, while algorithm information guarantees the shortest logical route. However, this is a very niche direction which makes it hard to find past research which studies this from the perspective of computation instead of physical concepts (like energy and entropy). The primary aim of this chapter is to explore the reverse relation, i.e., how AIT can be helpful in modeling QC.

The recent work from Marcus Mueller [162] based on John Wheeler's 'law without law' idea forms the primary motivation for this research. He provides a framework to formalize an operational formulation of quantum mechanics (QM) using only Solomonoff induction (and the assumption that the universe is computable). In this research, we propose its active version via an agent model. Agent models based on Solomonoff induction are studied under the theoretical discipline of universal artificial general intelligence. Here, we generalize the KL-KSA model to a quantum knowledge seeking agent (QKSA), using a density matrix formalism. Thus, this provides a general reinforcement learning perspective to the scientific modeling of quantum mechanics.

This chapter is based on the following:

- **Sarkar, A.**, Al-Ars, Z., Gandhi, H., & Bertels, K. (2021). QKSA: Quantum Knowledge Seeking Agent – resource-optimized reinforcement learning using quantum process tomography. *arXiv preprint arXiv:2112.03643.*
- **Sarkar, A.**, Al-Ars, Z., & Bertels, K. (2020). Quines are the Fittest Programs - Nesting Algorithmic Probability Converges to Constructors. *Preprints 2020,* 2020100584.

4.1. USE CASE MOTIVATION

Artificial general intelligence (AGI) is often referred to as 'the last invention humanity would need'. It aims to build a system that can perform general tasks across diverse disciplines on par with (or superior to) human abilities. While AGI was the original motivation within the field of artificial intelligence (AI), even today, AGI research is fragmented across approaches. Instead, artificial narrow intelligence (ANI) has had massive impact and success in the past decade, owing to the availability of computing power and novel neural network based learning models. Motivated by these successes, research on AGI is currently gaining traction.

The research focus of this dissertation, i.e., quantum computation (QC), and AGI have two important touch-points. QC's compute power might allow tractable general learning models. On the other hand, since AGI models should also be able to accomplish tasks that human scientists can perform, it should be capable of understanding and manipulating quantum information as well. In the previous chapter, we developed approaches to enable the former. In this chapter, we will focus on the latter aspect.

The application of automated scientific modeling is emphasized in this chapter. Theoretically rigorous AGI models based on universal computing best suit this application. Thus, we study these models and propose a generalization that takes into account quantum information. Our model, the quantum knowledge seeking agent (QKSA), also provides two conceptual improvements besides the quantum generalization. Firstly, it takes into account computational resource-bounded algorithmic complexity for a small subset of provided strategies. This allows our AGI model to be practically implemented. Secondly, the utility function of the reinforcement learning based QKSA is embedded within a self-replicating evolutionary code. This allows open-ended modeling over dynamically changing environments and computational resource trade-offs. While the primary contribution of this research is the QKSA formalization, a sufficiently large instantiation of this model can be applied for optimizing quantum computing algorithms.

4.2. ARTIFICIAL GENERAL INTELLIGENCE

The theoretical framework of intelligence helps us understand the capabilities and limitations of natural and artificial intelligence. Computational learning is being increasingly realized in diverse disciplines. This fascinating growth can, however only be sustained by achieving the following three crucial characteristics: (i) scalability - of computational resources allows applying the system to complex situations; (ii) explainability - focuses on human understanding of the decision from the learned solution; and, (iii) generality - involves using a single framework to address multiple scenarios. Despite the immense success of machine learning approaches, data-driven black-box models currently struggle to address these three aspects in tandem. In this research, we define a framework that addresses the requirements of these aspects simultaneously.

The holy grail of the field of automation is artificial general intelligence (AGI). While this was the eventual goal of even the founders of artificial intelligence, AGI has continually eluded computer scientists as a moving target. Encouraged by the recent achievements of intelligent systems, research on AGI is being revived and pursued from various directions [163], like, evolutionary approaches, neural networks, and symbolic logic.

The most theoretically advanced among these is universal artificial general intelligence (UAGI) [121]. It is a descriptive theory that is useful for studying super-intelligent AI without building one. The agent-environment paradigm of model-based reinforcement learning (RL) is best suited to mimic the interactive learning behavior of artificial and biological intelligence. Lately, the surge of interest in adaptive and autonomous devices has increased the prominence of RL methods beyond robotics and AI communities. UAGI based RL agents are concisely referred to as universal reinforcement learning (URL) agents. In this research, we examine policies of modeling the dynamics of an unknown environment by a URL agent.

URL agents have been instrumental in proving asymptotic optimal behavior in partially observable environments by merging theoretical concepts of decision theory, the notion of universal automata and algorithmic information theory (AIT). However, the dependence on AIT makes these agents generally uncomputable. While resource-bounded variants have been proposed, these models still remain intractable for real world applications. Moreover, the resource bounds introduce arbitrary hyper-parameters. To address this issue, we propose to use the idea of embedding RL agents within an evolutionary framework called EVO-RL [164] to guide the hyper-parameter tuning for a specific application scenario. In this work, we propose for the first time the idea of a resource-bounded EVO-URL. This is prompted by the suggestion of a UAGI system to eventually play the role of an autonomous scientist by recursive self-improvement (RSI) [165]. The RSI characteristics are ensured by embedding the agent's code within a mutating quine.

In defining AGI, the choice of a general environment is as crucial as that of a general learning strategy. Learning about a physical system by information exchange in its most general form should include classical, quantum, and relativistic scenarios. In this work, we address the first two cases by defining the environment as an unknown quantum process. The proposed agent uses quantum process tomography (QPT) as the general algorithm to learn and model the environment.

The major limitation of UAGI is the exponential scaling of the space of programs, which limits its applicability to very simple cases. To circumvent this, the agent policies in QKSA are chosen from a predefined pool of QPT strategies. This makes the agent policy computationally tractable as well as explainable, allowing a prescriptive theory of UAGI. The scalability is bounded by the exponential overhead of classical simulation. This computation cost can be considerably frugal for pragmatic approximation thresholds of classical shadows [166, 167] of quantum information. The proposed Quantum Knowledge Seeking Agent (QKSA), is an AGI framework based on resource-bounded EVO-URL. It models classical and quantum dynamics by merging ideas from AIT, quantum information, constructor theory, and genetic programming. Following the artificial life (or, animat) path to intelligence, a population of classical agents undergoes open-ended evolution (OEE) to explore pareto-optimal ways of modeling the perceptions from a quantum environment.

Operational effective theories of quantum mechanics have already been reconstructed based on classical information from measurements. A recent proposal [162] based on Solomonoff induction with the only assumption of computability from algorithmic information theory has been used to reconstruct predictive strategies for non-relativistic classical and quantum environments. This work is inspired by the 'law without law' idea from digital physics. Using QKSA, we extend this to allow the agents to develop strategies to choose the input states and measurement basis based on the 'participatory observer' notion. To complement this free choice of the agent, the learning goal is set to reward the predictive capability of environmental interaction while optimizing algorithmic and computational resources. QKSA does not assume quantum computational capability for the agent in line with the conventional qualia of human intelligence.

4.2.1. QUANTUM ARTIFICIAL INTELLIGENCE

Quantum artificial intelligence (QAI) is an umbrella term exploring the synergy between these two disciplines. It broadly entails either (i) using principles of quantum information and computation within artificial intelligence models or (ii) using artificial intelligence for processing quantum information, thereby advancing research in quantum technologies. In our past work [168], we have proposed general approaches and applications for the former, while in this research, we will focus on the latter case.

Quantum machine learning (QML) is a data-driven sub-field of QAI with a similar bidirectional synergy. Owing to the immense success of classical machine learning (ML), QML has been growing in popularity in recent years. In one direction, QML generalizes classical models like neural networks, clustering, regression, optimization, etc., to quantum information. These quantum algorithms running on quantum computers strive for a benefit in terms of runtime, trainability, solution quality, or memory space with respect to a classical ML approach. In the other direction, ML techniques are employed for optimizing and controlling processes in the development of quantum computers. These include control of the quantum system, routing and mapping of qubits, quantum error correction, etc.

Here we note five specific QML solutions which are closely related to QKSA. [169] uses restricted Boltzmann machines for learning quantum states and processes. The classical optimizer in quantum variational approaches has also been implemented as a reinforcement learner [170–173]. Variational quantum algorithms for reinforcement learning using evolutionary optimization [174] have also recently been proposed. These three implementations are based on data-driven neural networks. [175] implements a computer algorithm, Melvin, which finds new experimental implementations for the creation and manipulation of complex quantum states. However, the framework is not general for quantum information and is currently designed for optimizing experiments in quantum optics. Projective simulation (PS) [176–178] is a quantum reinforcement learning model that is the most similar in aim to QKSA. PS is a bio-inspired RL framework that allows the agent, based on previous experience, to project itself onto potential future situations using a stochastic network of clips called episodic and compositional memory. It aims to establish a general framework that connects embodied agent research with fundamental notions of physics. Despite the similarities with these approaches, these are not based on a universal computing model. They are either data-driven heuristics making them not easily explainable or applied to a specific context and is hard to generalize. Thus, they do not meet the definition of an AGI agent in the URL setting that we study in this research.

The proposed model of QKSA studies quantum information and computation via the lens of algorithmic information theory (AIT) [111] using reinforcement learning. QKSA is a generalization of existing classical URL models and thus allows evaluating the computational trade-offs of agency in the formalism of AIT. Recently, it was shown that it is possible to formulate any problem in AI as reinforcement learning [179]. Thus, our approach is inherently general for other QAI learning tasks. The QKSA framework consists of the classical agent, which performs the learning activity and the quantum environment, which defines the learning target. In this section, we first discuss the concepts used in defining a general agent and, thereafter, the concepts used in defining a general environment.

4.2.2. CLASSICAL AGENT

The design of the classical agent is based on three otherwise independent concepts from computer science. At the core is a generalization of a knowledge seeking agent. The hyper-parameters that define the resource constraints of the KSA are encoded as a gene. A population of agents uses genetic programming (GP) to evolve by mutation, thereby tuning these parameters. The KSA and GP are encapsulated within a self-replicating quine that allows recursive self-improvement. The background for these three characteristics of the proposed learning agent: knowledge seeking agents, genetic programming, and self-replicating programs, is explained in this section.

KNOWLEDGE SEEKING AGENTS FOR UNIVERSAL REINFORCEMENT LEARNING

Solomonoff's theory of universal inductive inference forms the theoretical basis of UAGI. It formalizes the two abductive heuristics that are used in scientific modeling, (i) Occam's razor or the principle of parsimony - i.e., when presented with competing hypotheses about the same prediction, select the model with the fewest assumptions, and (ii) Epicurus' principle of multiple explanations - i.e., retain all theories that are consistent with the observed data. In this theory, competing predictions are proportionally weighted by the size of the hypothesis that generates the prediction. This weight is called the algorithmic probability (AP). To estimate the hypothesis size (or algorithmic information), the environment being modeled is assumed to be computable by a universal Turing machine (UTM). The algorithmic complexity (AC) thereby defines the hypothesis size. While the exact values of AP and AC are uncomputable due to the halting problem, upper bounds can be estimated using techniques like the block decomposition method (BDM) [77]. The invariance theorem allows choosing any universal automata or language for the estimation. This adds a constant overhead based on the cross-compiler program length between the selected machine and the UTM.

UAGI is formulated in a general reinforcement learning (GRL) setting where the agent and environment interact in turns. At every time step, the agent supplies the environment with an action. The environment then performs some computation and returns a percept to the agent, and the procedure repeats. The environment is modeled as a partially observable Markov decision process. The agent cannot observe the underlying Markovian state directly but receives (incomplete and noisy) percepts through its sensors and thereby must learn and make decisions under uncertainty in order to perform well. The canonical model of UAGI is the AIXI model [121]. It is the active generalization of Solomonoff induction using Bellman's optimality equation.

Knowledge Seeking Agents [180] generalize the extrinsic reward function in AIXI to a utility function defined as information gain of the model. Thus, this collapses the exploration-exploitation trade-off to simply exploration, allowing agents to explore the environment in a principled approach. The goal of these agents is to entirely explore their world in an optimal way and form a model, and get reward for reducing the entropy (uncertainty) in its model from the two components: uncertainty in the agent's beliefs and environmental noise. A particularly interesting case is the KL-KSA [181], which is robust to stochastic noise (inherent in quantum dynamics) as the utility function is given as the Kullback-Leibler divergence or relative entropy.

Besides, since UAGI models are only asymptotically computable, it is not a pragmatic algorithmic solution to general RL and must be simplified in any implementation. In principle, there are an infinite number of programs that can be candidate models of the environment. Also, while evaluating, the programs can enter infinite loops. Thus, to circumvent these two issues, a modified (time and length bounded AIXI) agent called AIXI-tl [182] limits the length of the programs considered for modeling as well as assigns a timeout for computing the action. These resource considerations can similarly be translated to the KL-KSA case.

GENETIC PROGRAMMING FOR RESOURCE OPTIMIZATION

Complementary to the UAGI approach, evolutionary computation uses a different strategy based on biologically inspired models that evolve from a set of simple rules. It employs a population-based trial and error problem-solving technique for meta-heuristic or stochastic optimization. An initial set of candidate solutions is generated and iteratively updated. Each new generation is produced by selecting more desired solutions based on a fitness function and introducing small random mutations. The population mimics the behavior or natural selection and gradually evolves to increase in fitness. Different variants like evolutionary strategies, genetic algorithms, evolutionary programming, and genetic programming were developed to suit specific families of problems and data structures. There are other metaheuristic optimization algorithms that are also categorized as evolutionary computation, like agent-based modeling, artificial life, neuro-evolution, swarm intelligence, memetic algorithms, etc.

Genetic programming [183] is a heuristic search technique of evolving programs, starting from a population of (usually) random programs, for a particular task. Computer programs in GP are traditionally represented in memory as tree structures (as used in functional programming languages) which can be easily evaluated in a recursive manner. The fittest programs are selected for reproduction (crossover) and mutation according to a predefined fitness measure. Crossover involves swapping random parts of selected pairs (parents) to produce new and different offspring, while mutation involves the substitution of some random part of a program with some other random part of a program. GP has been successfully used for automatic programming, hyper-parameter optimization, machine learning, and automatic problem-solving engines. It is especially useful in domains where the exact form of the solution is not known in advance, or an approximate solution is acceptable.

QUINES FOR RECURSIVE SELF-IMPROVEMENT

The third characteristic that is crucial for the QKSA model is artificial life (alife). Alife examines systems related to natural life, its processes, and its evolution through the use of simulations with (soft) computer algorithms, (hard) robotics, and (wet) biochemistry models. The field of soft-alife was mainly developed using cellular automata [184], while neuro-evolution is another popular technique in use today. An idea foundational to alife is of a universal constructor. Universal constructor is a self-replicating automata developed 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. Additionally, the machine might have an overall operating system and extra functions as payloads. The payload can be very complex, like a learning agent or an instance of an evolving neural network. A quine is a program that takes no input and produces a copy of its own source code (and optionally other useful results) as its output. Thus, it is akin to the software embodiment of constructors. In principle, any program can be written as a quine, where it (a) replicates its source code, (b) executes an orthogonal payload that serves the same purpose as the original non-quine version. In § 4.5, we present a formal reasoning in support of modeling QKSA as a quine (a special subset of all programs possible on universal automata like Turing machine).

In summary, the crucial elements that will be used for defining QKSA are as follows. Firstly, the KSA types of URL are used. These reinforcement learning agents model the environment as programs on a universal automata. The programs output predictions of subsequent environmental percept when provided with the sequence of past actions and percepts. The current action is chosen based on the program which has the highest weight determined by (i) having a minimal length (and optionally, by other computational resources constraints like runtime) and (ii) having a high total expected information gain over the time horizon by the sequence of the chosen optimal actions and corresponding predicted perceptions. Secondly, GP is used for hyper-parameter tuning. The resource constraints are free hyper-parameters that evolve using mutation between generations of QKSAs. Thirdly, this EVO-URL framework is encapsulated within the payload of a quine. This consideration is independent of the quantization, and the reasoning can be extended to other models of AGI. It allows the QKSA to be a recursive self-improving agent.

4.2.3. QUANTUM ENVIRONMENT

The definition of a general environment and learning strategy is crucial for AGI research. In this section we present a brief overview of quantum information and computation, which generalizes classical and probabilistic information processing. We present quantum process and tomography as the corresponding general environment and learning strategy that will be used by QKSA.

DENSITY MATRICES

While pure quantum states can be written in terms of ket vectors $|\psi\rangle$, it cannot represent a mixed state, i.e. a statistical ensemble over a set $\{|\psi\rangle_k\}$ of N pure quantum states. Such states are described as a density matrix ρ, as the sum of the probabilities $0 < p_k \leq 1$ and $\sum_{k=1}^{N} p_k = 1$, multiplied by the corresponding projection operators onto certain basis states. It is defined as $\rho = \sum_{k=1}^{N} p_k |\psi\rangle\langle\psi|$. The bra vector is the adjoint (complex conjugate transpose) of the ket vector, i.e. $\langle\psi| = |\psi\rangle^{\dagger}$. Though the global phase of a quantum state is undetectable, i.e. $|\psi\rangle = e^{i\theta}|\psi\rangle$, a density matrix is unique, as the corresponding global phases of the bra and ket cancel out $\rho = |\psi\rangle\langle\psi|$. The corresponding evolution postulate by a unitary transformation U is $\rho' = U\rho U^{\dagger}$. A projective measurement of an observable M_m is given by the expectation value $Pr(m) = Tr(M_m\rho)$. The density matrix formalism deals with observable probabilities, whereas ket states deal with complex probability amplitudes. Statistics of quantum measurements can only estimate the density matrix instead of the state. Thus, we would use this within the QKSA formalism.

QUANTUM PROCESSES

A quantum process $\mathcal{E}$ that transforms a density matrix need not always be unitary. Given classical processes are often irreversible and include measurements, a quantum generalization includes unitary transforms (symmetry transformations of isolated systems), probabilistic logic, as well as measurements and transient interactions with an environment. Thus, quantum processes formalize the time evolution of open quantum systems. These are quantum dynamical maps, which are linear and completely positive (CT) map from the set of density matrices to itself. Typically they are non-trace-increasing maps and trace-preserving (TP) for quantum channels. For a quantum system with an input state ρ_{in} of dimension $n \times n$ and an output state $\rho_{out} = \mathcal{E}(\rho_{in})$ of dimension $m \times m$, we can view this system $\mathcal{E}$ as a linear superoperator mapping between the space of Hermitian matrices $\mathcal{E} : \mathcal{M}_{n\times n} \rightarrow \mathcal{M}_{m\times m}$. While ρ is an order 2 tensor (i.e. operator), acting on Hilbert spaces of dimension $D = 2^n$, $\mathcal{E}$ is an order 4 tensor specified by $D^4 - D^2$ parameters. Besides the superoperator, there are other equivalent [185] representations of quantum processes like Choi-matrix Λ, Kraus operators, Stinespring, Pauli basis Chi-matrix χ, Pauli Transfer Matrix, Lindbladian, etc.

For instance, the Choi matrix ρ_{Choi} is the density matrix obtained after putting half of the maximally entangled state $|\Omega\rangle$ through the channel $\mathcal{E}$, while doing nothing on the other half.

$$\Lambda = \sum_{i,j} \frac{1}{2^n} |i\rangle\langle j| \otimes \mathcal{E}(|i\rangle\langle j|)$$

$$\rho_{Choi} = \Lambda(|\Omega\rangle\langle\Omega|)$$

Thus it requires $2n$ number of qubits, but since the input state is fixed, this effectively involves performing a quantum state tomography (QST) on this larger space instead of QPT, reducing the overall number of trials. The evolution of a density matrix with respect to the Choi-matrix is given by:

$$\rho_{out} = \mathcal{E}(\rho_{in}) = \mathrm{Tr}_1((\rho_{in}^{T} \otimes I)\rho_{choi}))$$

where Tr_1 is the partial trace over subsystem 1. As a result of the Choi-Jamiolkowski isomorphism, the Choi matrix Λ characterizes the process $\mathcal{E}$ completely. This forms the basis of the channel-state duality between the space of CP maps and the space of density operators.

QUANTUM TOMOGRAPHY

Characterization of quantum dynamical systems is a fundamental problem in quantum information science. Several procedures that have been developed that achieve this goal are called quantum process tomography (QPT). Some examples of QPT techniques are: standard quantum process tomography [186], entanglement-assisted quantum process tomography (EAQPT), direct characterization of quantum dynamics, compressed-sensing quantum process tomography, permutation-invariant tomography, and self-guided quantum process tomography [187] Each QPT technique has a different experimental setup and computational resource requirements. An exhaustive survey of all techniques considering an inclusive figure-of-merit with respect to resources and their trade-offs has not been undertaken previously. [188] provides a good overview of some of the most used techniques and comparison in terms of the resources of the Hilbert space, input state complexity, required measurement, and required interactions. In the proof-of-concept implementation of QKSA, we will use EAQPT for the experimental results. EAQPT is based on the Choi-Jamiolkowski isomorphism, as it uses QST to reconstruct the Choi density matrix of the quantum process.

A more recent development towards the limits of quantum tomography is based on the classical shadow of states [189, 190] and processes [166, 167]. Shadow tomography aims to extract essential information about a state/process with only polynomially many measurements. These form a good set of candidate QPT algorithms that can be used to explore the resource trade-offs for limiting cases.

As a summary, within the QKSA formalism, QPT reconstruction algorithms form the space of programs that are evaluated by the agent as candidates for the modeling of the environment. Given computational resource limitations, QKSA can automatically discover the optimal strategy in the available pool of QPT algorithms. In canonical UAGI formalism, the pool of programs is drawn randomly from a prefix-code for a universal automata. However, the space of programs grows exponentially, limiting their applicability. We restrict this space to a constant number of predefined algorithms. Intuitively, a QPT algorithm will perform better in predicting a quantum environment than a random program. Thus, it allows us to apply the tools of AIT in a practical setting where available expert knowledge can be embedded within the agent, making it tractable.

4.3. QUANTUM KNOWLEDGE SEEKING AGENT

QKSA extends the universal reinforcement learning (URL) agent models of artificial general intelligence to quantum environments. Despite its importance, few quantum reinforcement learning models exist in contrast to the current thrust in quantum machine learning. QKSA is the first proposal for a framework that resembles the classical URL models. Similar to how AIXI-tl is a resource-bounded active version of Solomonoff universal induction, QKSA is a resource-bounded participatory observer framework for the recently proposed algorithmic information-based reconstruction of quantum mechanics. The utility function of a classical exploratory stochastic Knowledge Seeking Agent, KL-KSA, is generalized to distance measures from quantum information theory on density matrices. Quantum process tomography (QPT) algorithms form the tractable subset of programs for modeling environmental dynamics. The optimal QPT policy is selected based on a mutable cost function based on algorithmic complexity as well as computational resource complexity. Instead of Turing machines, we estimate the cost metrics in a high-level language to allow realistic experimentation. The entire agent design is encapsulated in a self-replicating quine which mutates the cost function based on the predictive value of the optimal policy choosing scheme. Thus, multiple agents with pareto-optimal QPT policies evolve using genetic programming, mimicking the development of physical theories, each with different resource trade-offs. A proof-of-concept is implemented and available as an open-sourced software. Besides its theoretical impact, QKSA can be applied for simulating and studying aspects of quantum information theory like control automation, multiple observers, course-graining, distance measures, resource complexity trade-offs, etc. Specifically, we demonstrate that it can be used to accelerate quantum variational algorithms, which include tomographic reconstruction as its integral subroutine.

In this section, the features and the formal framework of the QKSA is presented by defining the parameters within an implementation that captures the agent scheme discussed so far. Thereafter, an execution procedure and the framework are outlined.

4.3.1. Characteristic features of QKSA

The main distinguishing features of QKSA with respect to other approaches are presented in this section.

Classical observers in a quantum world

Similar to UAGI, in digital physics [191], the universe is modeled as a vast (quantum) computation device or as the output of a deterministic or probabilistic computer program. Information is increasingly put into the central stage in physics, especially in reconstructing theories like quantum mechanics from general principles [192–194] as well as its physical nature [195]. John Archibald Wheeler [196] popularized this idea as 'it from bit'. This meant that every item of the physical world has at its bottom an immaterial source and explanations of what we call reality arise from the posing of yes-no questions and the registering of equipment-evoked responses. Quantum information theory as a generalization of Boolean logic is used by Seth Lloyd [197] to extend this principle, with the evolution of the universe as an ongoing quantum computation, with the fundamental laws of physics constituting the program. Wheeler asked the question of the possibility of the existence of an ultimate law of physics, from which everything that is knowable about the material world can be deduced. This idea has been coined as 'law without law' [198]. If such a principle does not exist, it would mean that certain aspects of the natural world are fundamentally inaccessible to science. Instead, the existence of such a unified law would have to explain its own origin and preferential bias. So, paradoxically, the ultimate principle of physics cannot be a 'law' (of physics), hence the expression. Thus epistemological assumptions of how physical theories are formed and verified become imperative, removing physics as the science of 'what is' to that of 'what we observe'.

The recent work from Markus Müller [162] is central to the ideas developed in this research. It claims that given a complete description of the current observer state x, it is possible to predict what state y the observer will subsequently evolve to using $P(y|x)$ based on Solomonoff's algorithmic probability, universal prior, and universal induction.

This currently encompasses classical (non-relativistic) and quantum physics and can be used to reconstruct an operational theory based on this assumption of the world being computable and there are no super-Turing physical processes. Note this is entirely an epistemic derivation of the laws of physics based on information axioms and focuses on using the theories to predict the results of future interactions instead of assuming any ontological interpretations of the model.

In this research, we extend the research from [162] to an implementation of a framework to allow experimentation. While doing so, we narrow down the specifics of the original ideas. The primary objective of AGI models like QKSA is to mimic human behavior to form an explainable hypothesis about the environment. Semantic explanation based on human qualia is represented in terms of classical information. This does not restrict representing quantum information, as using the standard formalism, we can represent quantum information using a worst-case exponential amount of real-valued classical information. Thus, the QKSA specifically models classical observers in a quantum world and still recover and learn features that can help us form hypothesis and predict the environmental dynamics.

Quantum process tomography as a general modeling technique

Extending Müller's idea to the Church-Turing-Deutsch thesis, the program (that the Solomonoff induction uses) is basically an efficient quantum computing simulator, given a classical computing substrate. This can also be a programmable quantum simulator [199] given a quantum computing substrate. A model of the environment (universe) is created from the agent's (observer's) perspective, representative of the black-box input-output behavior of the environment. Given knowledge of the environmental dynamics, it is possible to create the corresponding classical model (e.g., a Grover search simulator). However, for unknown environments, the general technique is to perform process tomography; thus, that is the general modeling algorithm we would focus on. For the rest of this article, we consider the general case of quantum environments. A classical environment can be efficiently mapped to a corresponding quantum environment.

For the general quantum case, what kind of algorithms would execute for predicting the next observer state using Solomonoff universal induction? Given that it is possible to simulate quantum physics on a classical simulator (albeit by incurring exponential resource cost), the good predictor will be a quantum process tomographic reconstruction based on the previous observer states. Thus, an agent trying to derive the information-based operational laws of quantum physics would converge to a QPT algorithm as the best predictor for subsequent environmental percepts. And thus, to define a tractable formalism, we can consider the subset of all QPT algorithms instead of the entire space of programs for the universal automata.

Participatory observer as a reinforcement learning agent

Consider the phase before the process matrix has been reconstructed to a certain degree of precision (i.e., before an informationally complete history of observations is registered). In this phase, the participatory observer can choose an action like a UAGI agent based on the process matrix. The subsequent perception will be based on both the chosen action and the environment. Thus, this phase is not fully modeled by Solomonoff's induction. In the reinforcement learning setting, the next observer state is based on both the current observer state (that defines the memory of previous observations and the current action based on the QPT scheme) as well as the part of the environmental dynamics that has not yet been learned.

Given a complete description of the environmental dynamics already learned and encoded within the observer state x, it is still not possible for the agent to perfectly predict individual measurements. However, the statistical distribution of measurement results in any chosen basis can be predicted by the agent. Thus instead of predicting individual perceptions, a quantum UAGI agent can only predict the expected probability of a measurement. This is inherently dependent on the choice of a measurement basis from the agent. This is called the participatory observer principle, which states - physical things are information-theoretic in origin, and this is a participatory universe where the interactions define the reality we perceive. QKSA, in effect, learns efficient strategies for observer participancy for modeling quantum environment.

Open-ended evolution of Pareto-optimal computational resources

The algorithmic probability of the program is used as a weight for the chosen action and the reward in UAGI. However, this also makes such models impractical due to the uncomputability of mathematical objects in algorithmic information like algorithmic probability and algorithmic complexity. Thus, pragmatic implementation of these models like AIXI-tl, MC-AIXI$_{\text{(FAC-CTW)}}$ and UCAI [200] bounds the program length and runtime per step to explore a subspace of promising hypotheses that models the interactive behavior registered till the current time step. Three issues arise with this approach:

1. The bounds introduce heuristic hyper-parameters that depend on the available computational resources. Thus it becomes difficult to select an appropriate value to apply the model for a given use-case.
2. The bounds sharply cut off models beyond the specification while keeping the weight for the models within the specification unaffected. So a model that performs well but just lies beyond the defined bound may be unreachable.
3. It is possible to trade off these resource bounds with other computational resources, like additional memory.

Using the QKSA platform, it is possible to investigate these issues. In the framework we consider five computational resources together called the least metric, as an acronym for (program/hypothesis) $\underline{\text{l}}$ength, (compute) $\underline{\text{e}}$nergy, $\underline{\text{a}}$pproximation, (work memory) $\underline{\text{s}}$pace and (run) $\underline{\text{t}}$ime. Similar algorithmic observables have been suggested in [201]. We provide estimation techniques of the least metric for each based on state-of-the-art algorithmic information research and general practices in computer engineering. The estimation technique, however can easily be redefined by the user. These estimated metrics are used in a two-fold way. Firstly, it is used to qualify the hypothesis for consideration based on an upper bound for each of the five metrics individually. This is dictated by the available computational resource and is similar to the resource-bounded UAGI models. These bounds can be included in the list of evolving hyper-parameters to allow QKSA to mutate and adjust autonomously to the available computational resource. Thereafter, the metrics for a valid hypothesis are fed to a cost function (a genetic program) that outputs a single positive real value which is used as the weight for the hypothesis in the semi-measure instead of only the length, as in algorithmic probability. We call this the 'computational action' as a parameter to optimize the Lagrangian dynamics within computational space-time.

No unifying cost function exist that can serve as a metric to trade-off bounds on resources (like space, time, approximations). This depends closely on the policy of the agent. For example, a physicist might choose to use simpler Newtonian mechanics instead of complex relativistic mechanics for modeling where the approximations are acceptable. Thus, instead of a single metric, a pareto-optimal frontier on the least metrics maps to models and algorithms that can be used to predict the environment dynamics. Various research has explored this frontier, considering a few of the least metrics. For example, Bennett's logical depth and Schmidhuber's speed prior trades off time-length; Wolpert's research deals with the thermodynamic complexity of Turing machines; or look-up tables that trades off time-space.

QKSA holistically explores these trade-off via the GP function. The five estimates of the least metrics are input to a cost function. The cost function itself is a gene represented as a program tree with the leaf nodes as the metrics or constants, and the internal nodes are from a set of basic arithmetic functions (addition, multiplication, square root, logarithm, etc.). Once QKSA learns an environment optimally or completely fails to learn the environment (i.e., when the learning rate stabilizes), the QKSA self-replicates by invoking the quine functionality. The child QKSA has the same source code as the parent, except for a mutation on the cost function that modifies the weights and structure embedded via the cost function gene. Thus the open-ended evolution of the pareto-optimal manifold converges on QPT algorithms which fit well with the available computational resource. The parent QKSA perishes if the prediction of the model fails persistently (i.e., when the rate stabilizes as the strategy fails to learn) or continues to correctly predict environmental interaction and can be inspected to obtain the cost function.

UTILITY FUNCTION AS QUANTUM COMPLEXITY DISTANCE

The learning process in RL is guided by a reward function assigned by the environment as part of the perception. While this is trivial to define for game environments, it is difficult to define for modeling environment dynamics without introducing another third-party evaluating agent. To circumvent this, we consider the generalization of rewards as utility computed by knowledge seeking agents, instead of an external input. The utility is a metric estimated internally by the agent, based on a self-defined distance measure in the space of percepts. As already discussed, due to the inherent randomness of quantum measurements, it is not possible to predict individual measurements in an arbitrary basis even with full knowledge of the system. Thus, the metric is evaluated on the stochastic distribution of percepts. The process matrix reconstructed by the QPT algorithm from the already known history of action and corresponding percept is called ρ^{t}_{Choi}. An update of this matrix $\rho^{\prime}_{Choi}$. is predicted based on the current chosen action a_t and the corresponding prediction of the percept $e^{\prime}_{t}$. The actual update ρ^{t+1}_{Choi} is however based on the actual perception from the environment e_t. The distance measure between these two updates is the utility. Once the process matrix is fully learned, this distance will converge. Thus, QKSA is a generalization of KL-KSA for density matrices.

Unlike classical probability distribution, there are many measures of quantum distances, each with its own application advantage. The QKSA framework allows the user to select a distance metric as part of the experimental setup. The current version provides the following distance metrics, Hamming distance, KL divergence, trace distance, Hilbert-Schmidt norm, and Bures distance (fidelity). Users can also define their own custom distance measure. A future extension would provide diamond distance, Hellinger distance, quantum Kolmogorov complexity, quantum relative entropy, Rényi divergence, Bhattacharyya distance, and quantum complexity action [202].

4.3.2. DEFINITIONS OF PARAMETERS

The standard formalism of reinforcement learning includes:

- t_p is the number of time steps in the past that is considered by the agent at each point in time. In AIXI, this considers all steps since the inception of the agent. For pragmatic implementations and in QKSA, it is typically a sliding window of a few steps in the recent past based on the available total memory of the agent.
- $t_f \in \{1, \infty\}$ is the number of time steps that the agent predicts in the future or the remaining duration the agent is run. It is also called the horizon. In the limiting case, the number of steps for asymptotic convergence to the optimal strategy is infinity. For QKSA, we will consider only 1 step in the future, but the implementation is generic and can be extended to any number of steps.
- $a_t \in \mathcal{A}$ is the chosen action from the action space at time step t.
- $e_t \in \mathcal{E}$ is the perception recorded by the agent at time step t from the percept space.
- $e_t^{'} \in \mathcal{E}$ is the prediction of the perception e_t made at time step $t-1$.
- $\lambda^{e_t^{'}} \in \{0,1\}$ is the probability of the prediction $e_t^{'}$ made at time step $t-1$.
- h_t is the sequence of the history of actions and perceptions up to time step $t-1$. It implemented as a ring buffer of $h_t = a_{t-t_p} e_{t-t_p} \dots a_{t-1} e_{t-1}$
- ρ_t is the hypothesis or model of the environment generated by processing the history h_t by a candidate QPT reconstruction algorithm. It is typically a Choi matrix of the learned environmental quantum process.
- p_{qpt} is the QPT program that is executed on the defined computational model C. It is capable of, (i) generating a tomographic reconstruction ρ_t of the environment given the history h_t, (ii) provide an expectation value of a prediction $e_t^{'}$ given a ρ_t and a_t.

The least metric defines the bounds on the hypothesis space and the relative weight assigned to each considered hypothesis. It takes into account the 5 cost metrics of program length, thermodynamic cost, approximation, space/memory, and run-time. The hypothesis-space of QPT is bounded by the 5 least$_{max}$ hyper-parameters. All trial hypotheses must lie within the bounds of all 5 parameters. Once a trail hypothesis is admitted based on the least$_{max}$ bounds, the estimate of the 5 cost parameters least$_{est}$ are combined to form a single indicative metric of the fitness of the hypothesis. Each parameter has an associated weight or scaling factor w_{least}. The cost function defines the equation to combine the least$_{est}$ and w_{least}, and is subject to evolution.

- d refers to the data on which the least metric is evaluated. It consists of the history h_t and the QPT algorithm p. The cross-compiler description length between the chosen automata C (a Python compiler) and the canonical UTM is assumed to be constant and ignored for relative evaluation.

- l_{max} is the maximum length of p that is considered.
- e_{max} is the maximum energy cost of executing p for the functions discussed above.
- a_{max} is the maximum approximation threshold used by p for the functions.
- s_{max} is the maximum space or working memory that p can use during execution. It typically includes the h_t as well.
- t_{max} is the maximum execution time for p before it generates the output for the functions.
- l_{est} is an estimate of the length of p that is considered, that outputs d. A rough estimate can be arrived at by the bit length of the QPT program. Lossless compression or BDM [77] can also be used for a more tight estimate.
- e_{est} is an estimate of the energy cost of executing p. Research into this aspect is scarce, especially for high-level programs. The recent proposal on the thermodynamic Kolmogorov complexity [203] needs to be explored further for estimating the energy cost. It can also be externally estimated by the energy consumption of the computational automata C.
- a_{est} is an estimate of the deviation arising from approximations made by the program p.
- s_{est} is an estimate of the space or working memory that p uses during execution.
- t_{est} is an estimate of the execution time for p before it generates the output.
- $w = \{w_l, w_e, w_a, w_s, w_t\}$ is a set of associated weights for each of the least metrics.
- c_{least} is the cost function that takes in the 5 least metrics and a weight for each metric and calculates a cost based on the evolving gene.
- m_c is the mutation rate of the cost function c_{least}.
- F is the set of functions allowed in the cost function c_{least} and typically includes standard operations like addition, multiplication, exponentiation, logarithm, etc.
- c_{est} is the estimated cost based on the estimated least metrics and the cost function. This is the generalization of the program length as used in UAGI.

The parameters for the quine define the learning progress and when the agent self-replicates. Replication is triggered based on the fitness of the hypothesis based on the predictive capacity over time.

- Δ is a distance measure (e.g., Hamming distance, trace distance) defined between process matrices ρ.
- u'_t is the predicted utility between the predicted update to the process and the current learned process. It is the relative distance using $u'_t = \Delta(\rho'_{t+1}, \rho_t)$.
- γ_t is the discount that is proportional to the time span t between the predicted reward/utility step and the current time step. It depends on the dynamic and episodic nature of the environment. For time steps further in the future, prediction penalties can be scaled down. For episodic environments, the value is 1 for the next time step and 0 otherwise.
- $R_t = \sum_{e'_t \in \mathcal{E}} \lambda_t \cdots \sum_{e'_m \in \mathcal{E}} \lambda_m \sum_{k=t}^{m} \gamma_k u'_k$ is the cumulative discounted return at time step t. Note that since the QPT algorithms chooses predictions probabilistically, the weighted summation over the sequence of predictions needs to be considered.
- u_t is the actual utility between the interaction steps. It is the relative distance $u_t =$

$\Delta(\rho_t, \rho_{t-1})$.
- $K_t = u'_t - u_t$ knowledge at time step t.
- K_R is the knowledge threshold for reproduction. If $K_t < K_R$, the agent self-replicates with mutation in its hyper-parameters.
- K_D is the knowledge threshold for death. If $K_t < K_D$ the agent halts (dies).

4.3.3. AGENT FORMALISM

The main advantage of using universal reinforcement learning is that it allows us to define the learning model mathematically. In this section, we start from the formalism of the classical KL-KSA and elucidate the changes that lead to the QKSA formalism.

For simplicity, the action and percept spaces are assumed to be stationary (i.e., time-independent and fixed by the environment) and countable (although most results generalize to continuous spaces). The agent is formally identified by its policy π, which in general is a distribution over actions for the current step, conditioned over the history, denoted by $\pi(a_t|h_t)$. The environment is modeled as a distribution over percepts, $\nu(e_t|h_t a_t)$. A rational agent based on Von Neumann-Morgenstern utility theorem strives to maximize the expected return, called the value. The value achieved by a policy in an environment given a history is defined as: $V^\pi_\nu(h_t) = \mathbb{E}^\pi_\nu[R_t|h_t]$. This can be expressed recursively, as the Bellman optimality equation,

$$V^\pi_\nu(h_t) = \sum_{a_t \in \mathcal{A}} \pi(a_t|h_t) \sum_{e_t \in \mathcal{E}} \nu(e_t|h_t a_t)[\gamma_t r_t + \gamma_{t+1} V^\pi_\nu(h_{t+1})]$$

AIXI-based models use Solomonoff's universal prior for mixing over the model class $\mathcal{M}$ of all computable probability measures using the Kolmogorov complexity of the environment, $w_\nu = 2^{-l(\nu)}$. The environment is usually modeled as programs on a UTM, denoted as U, typically a monotone TM with 3 tapes, for input (perception), working, and output (action). Thus, $\xi(e_t|h_t a_t) = \sum_{\nu \in \mathcal{M}} w_\nu \nu(e_t|h_t a_t)$ and the optimal policy maximizes the ξ-expected return, i.e. $\pi^{\text{KL-KSA}} = \arg\max_\pi w_\nu V^\pi_\xi$. Distributing the max and $\sum$ in the recursive equation yields the canonical expectimax equation as,

$$a_t = \arg \lim_{m\to\infty} \max_{a_t \in \mathcal{A}} \sum_{e_t \in \mathcal{E}} \dots \max_{a_m \in \mathcal{A}} \sum_{e_m \in \mathcal{E}} \sum_{k=t}^{m} \gamma_k u_k \sum_{q:U(q;a_{<k})=e_{<k*}} 2^{-l(q)}$$

In the case of KL-KSA, the reward for AIXI is generalized to the utility given by

$$u(e_t|ae_{<t}a_t) = Ent(w_\nu|ae_{<t+1}) - Ent(w_\nu|ae_{<t}a_t)$$

The first change is to restrict the search space of programs p to quantum process tomography algorithms, denoted as p_{qpt}. Strictly there is no need to specialize the search to this subspace of programs. Searching over the full space of programs would lead to higher rewards for QPT algorithms owing to their predictive capability and thereby select actions based on this subspace. However, since we are interested in a pragmatic implementation, searching the full space of programs is intractable even for very modest cases. It is important that the QPT algorithm reconstructs and outputs a process representation ρ_t instead of the prediction of the next perception. This is imperative due to the stochastic nature of individual quantum measurements and the calculation of the utility.

The second change is to replace the length estimate of the $2^{-l(p)}$ factor from the algorithmic probability with the estimate of the evolving cost function c_{est}. The cost function is denoted by c_{least}, i.e. $c_{est} = c_{least}(p_{qpt})$. Thus, the learning part of the equation is:

$$\boxed{a_t^{QKSA} = \arg \lim_{m\to\infty} \max_{a_t\in\mathcal{A}} \sum_{e'_t\in\mathcal{E}} \lambda^{e'_t} \dots \max_{a_m\in\mathcal{A}} \sum_{e'_m\in\mathcal{E}} \lambda^{e'_m} \sum_{k=t}^{m} \gamma_k u'_k \sum_{\substack{p_{qpt}:U(p_{qpt};h_k)=\rho_k \\ p_{qpt}:U(p_{qpt};\rho_k;a_k;e'_k)=\lambda^{e'_k}}} 2^{-c_{least}(p_{qpt})}}$$

The third change is to define the utility function as a quantum distance measure on the space of quantum processes ρ defined as the density matrix in the Choi process matrix representation. A higher predicted utility indicates that the current estimate of the process will be updated more significantly based on the perception, thus, a potential knowledge gain for choosing that action. These relationships are shown in Figure 4.1.

$$\boxed{u'_t = \Delta(\rho'_{t+1}, \rho_t) = \Delta(U(p_{qpt}; h_t; a_t; e'_t), U(p_{qpt}; h_t))}$$

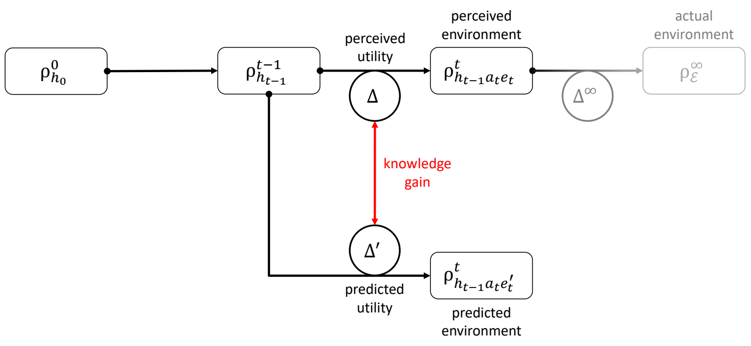


Figure 4.1: QKSA knowledge gain.

Episodic environment

While the infinite horizon is used for proving asymptotic optimality, a finite horizon is required for any pragmatic implementation. Since the QPT environment is episodic, i.e., the environment is reset after each interaction cycle, a horizon of 1 step captures the highest possible level of temporal dependency. The simplification of the policy for 1-step horizon, i.e. $k = m = t$ is:

$$a_t^{QKSA} = \arg\max_{a_t\in\mathcal{A}} \sum_{e'_t\in\mathcal{E}} \lambda^{e'_t} \Delta(\rho'_{t+1}, \rho_t) \sum_{\substack{p_{qpt}:U(p_{qpt};h_t)=\rho_t \\ p_{qpt}:U(p_{qpt};\rho_t;a_t;e'_t)=\lambda^{e'_t}}} 2^{-c_{least}(p_{qpt})}$$

Let us understand this simple case in more depth. At each step, the QKSA algorithm consists of two phases, learning, and evolution:

- In the learning phase:
  1. A pool of QPT algorithms is inspected.
  2. Each QPT algorithm is used to reconstruct the unknown environment as a process matrix based on the history of actions and perceptions.
  3. This reconstruction process is used to estimate the resource cost of the QPT algorithm.
  4. The process matrix is then used on all possible actions the agent can take at this step.
  5. For each such action, the process matrix predicts a distribution of perceptions. Thus, for each action-prediction pair, the process matrix generates a probability for the prediction using the QPT algorithm.
  6. Each predicted perception would lead to a predicted update for the process matrix.
  7. These predicted process matrices are compared with the current process matrix to generate the predicted utility of the corresponding action-prediction pair.
  8. These predicted utilities are weighted by the probability of that specific prediction that led to the predicted utility.
  9. These utilities for a specific action are accumulated as the utility for the action for all possible predictions of perceptions.
  10. This sum of utility for an action is weighed by the resource cost of the QPT algorithm used for modeling and prediction.
  11. The action that maximizes this weighted value is chosen as the action for the current step.
- In the evolutionary phase:
  1. The utility is used to calculate the return over the number of prediction steps based on the weights for each prediction used to calculate the utility.
  2. The total return is the learning gradient.
  3. If this return is below a threshold, the agent reproduces by mutating the cost function and self-replicating.
  4. Alternatively, if the return is above a threshold, the agent dies. There is also a maximum limit on the number of interaction steps and the number of reproductions, after which the agent halts.

4.3.4. EXECUTION PROCEDURE

The QKSA framework, as shown in Figure 4.2, consists of 5 major blocks: environment, a pool of QPT algorithms, LEAST metrics cost estimators, choice of distance measure, and the QKSA hypervisor. The execution procedure and the interaction between these blocks are explained in this section.

The environment is defined by the user as a Qiskit quantum circuit. The QKSA also allows probabilistic mixtures of quantum circuits and partially observable environments. Currently, only episodic environments are considered. Thus, each cycle of agent interaction resets the environment based on the focus of QPT. In non-episodic environ-

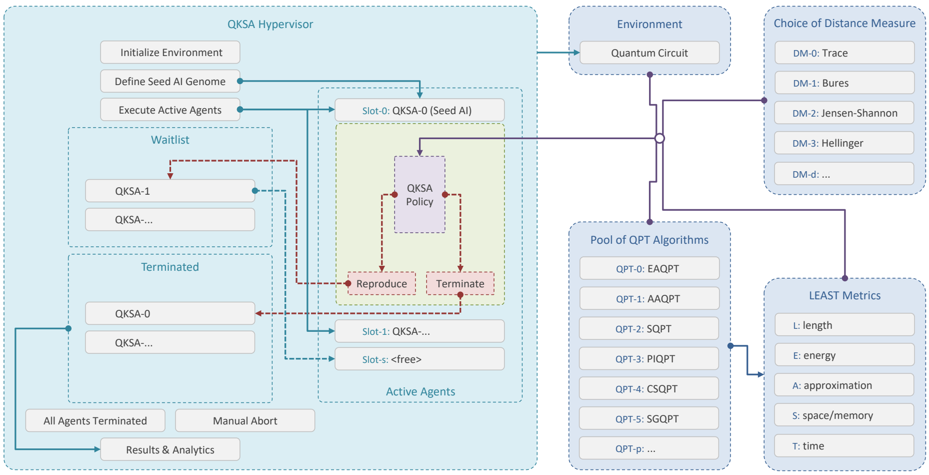


Figure 4.2: The QKSA framework.

ments, the Holevo bound restricts the total classical bits of information that can be extracted, limiting the applicability in model-based reinforcement learning. The environment is shared between the agents. Each agent can choose to measure only a part of the shared environment, and thus can be used for studying non-local strategies. The environment also defines the set of actions $\mathcal{A}$ and perceptions $\mathcal{E}$ that can be used by the agent for interaction. This set is defined automatically from the number of qubits used to define the environment. The user can, however modify and restrict the set based on the intended purpose, e.g., only Z-basis measurements are required for studying quantum versions of classical logic like a quantum adder.

The second block consists of a pool of QPT algorithms. Each algorithm is capable of taking as input the history h_t and output the environment model ρ_t. Any new QPT strategy can be coded and added to the pool as a black-box algorithm as long as this criteria is met. Note that initially, the history is an empty sequence. This corresponds to a maximally mixed density matrix.

Each QPT algorithm can also be evaluated for the LEAST metrics based on the cost estimators. The cost estimators use both online and offline methods to estimate the cost. For example, the length and approximation estimate of the QPT algorithm can be directly inferred from inspecting the program code, while the run-time is estimated while the optimal action is being evaluated.

The framework also offers a pool of distance metrics Δ between quantum density matrices ρ. The goal of QKSA is different from QPT for device characterization as the environment is unknown. Thus, the distance between the current model and the actual model cannot be calculated. Instead, the metric measure between the predicted model update and the actual update is used to infer the learning gradient. From our experiments, we provide a set of metrics that have a monotone behavior and are a distance measure for quantum processes. However, there are many distance measures, and the user can choose a specific default measure or let each agent randomly choose one during

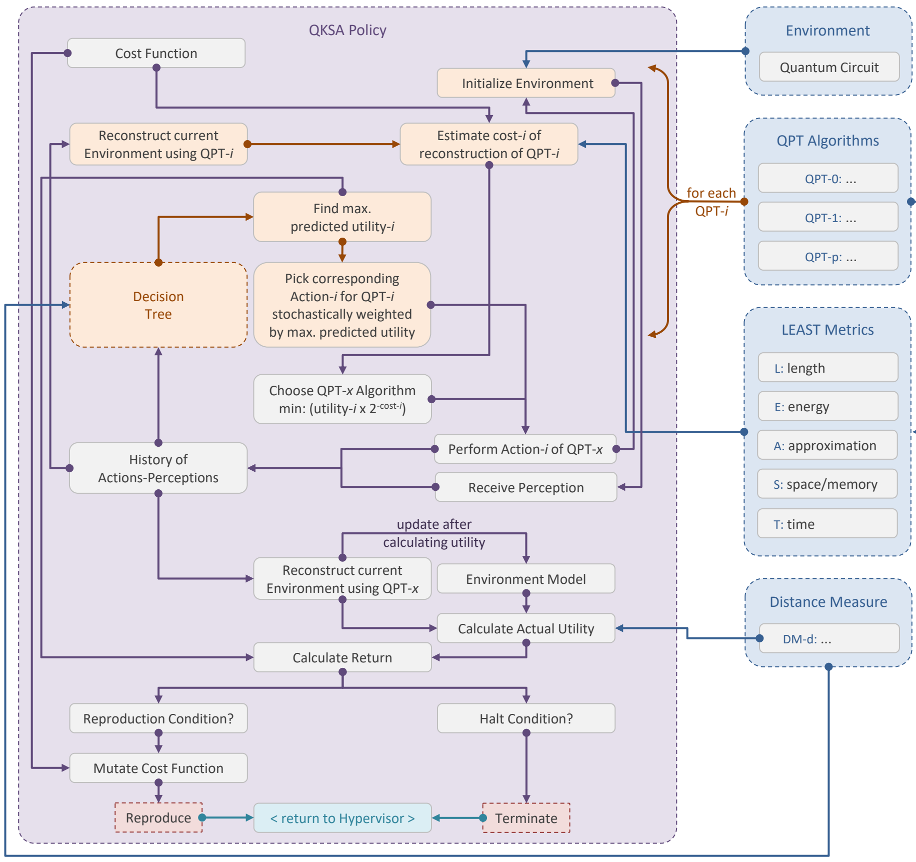


Figure 4.3: The QKSA policy.

the initialization.

The last module is the hypervisor that encapsulates the QKSAs. The seed QKSA constitutes the minimal implementation of the QKSA. This agent is instantiated by the QKSA hypervisor and added to the active pool of agents. Thereafter, the hypervisor executes each active agent, either in parallel or by dovetailing. Each agent learns the environment based on its own policy. When the learning converges, the agents reproduce by mutating their policy. The new agents are added to the waitlist and are automatically instantiated by the hypervisor when computational resources are available. Eventually, the agents completely learn the environment (or the maximum lifetime limit is reached). Then the agent is terminated. The user can also manually terminate all active agents. Thereafter, the learning results of each active/terminated agent are displayed for analysis.

In the following, the QKSA policy is explained in further depth. This is depicted in Figure 4.3. Each QKSA has its own Cost Function, which is its part of the mutable genome. Other parameters are passed to the QKSA from the hypervisor (for the seed

QKSA) or the parent via immutable genes in the remaining part of the genome. These parameters are used to initialize the agent. At this stage, the quantum circuit for each available QPT algorithm is also created. This might involve additional initialization circuits for the QPT strategy (e.g., entangling ancilla for EAQPT).

Then for each QPT algorithm, the best action is evaluated based on the corresponding predicted utility. This utility is based on the distance between the current environment model and the predicted environment model based on the chosen action and probabilistic perception. The LEAST cost for the QPT is also evaluated while the reconstruction is done. After this is done for all QPT algorithms, the weighted (by the cost) maximum utility is used to pick a QPT strategy.

The action of the corresponding chosen QPT is performed, and the perception is received from the modified environment of the specific QPT algorithm. This is used to calculate the actual utility as well as update the current model of the environment and the history.

The difference between the predicted and the actual utility is the return. This value is used to determine the learning progress and trigger the reproduction or termination of the agent.

The learning routine first evaluates the cost of each QPT strategy based on the cost estimators for choosing the current optimal action. QPT strategies that are beyond the allowed threshold are filtered out. A weighted selection is made for a specific QPT algorithm based on the cost of the remaining strategies. This defines the actual action to the environment and the prediction from the chosen QPT algorithm. The actual perception from the environment is used to update the model and calculate the utility for the current prediction based on the distance metric for the agent. The utility over the past steps is evaluated to assess the return. When the utility falls below a threshold, the return is used to determine the fitness of the agent. If the agent is fit, it replicates with a mutation on the cost function, otherwise it halts. In the standard default setup, the only difference between agents is the cost function. The replication is carried out by invoking the mutating quine subroutine. The new program file for the child QKSA is automatically instantiated by the hypervisor. The parent quine after reproduction continues predicting the environment and reproducing. An upper bound on the number of replications is set after which the agent is archived by the hypervisor.

4.3.5. EXPERIMENTAL RESULTS

A full proof-of-concept of the discussed QKSA framework is implemented on Python and Qiskit. It is available as an open-sourced software on GitHub at the following link: https://github.com/Advanced-Research-Centre/QKSA. In this section, we present an initial experiment that demonstrates the features of QKSA as presented in the previous sections.

In this experiment, we consider the choice between two QPT strategies, both EAQPT. QPT-0 has an approximation of 5 decimal places while using 16384 steps of history. QPT-1 has an approximation of 8 decimal places however uses only 8192 steps. Since they are the same algorithm, the program length variance is negligible. The time to reconstruct for QPT-0 is more because of the history (though the coarser approximation reduces it slightly). Given such a description, it is not immediately clear which QPT would work best to reconstruct and model a given quantum environment. It also depends on how simple/complex the environment is (e.g., if the additional decimal places have useful information). In such situations, QKSA can be readily applied.

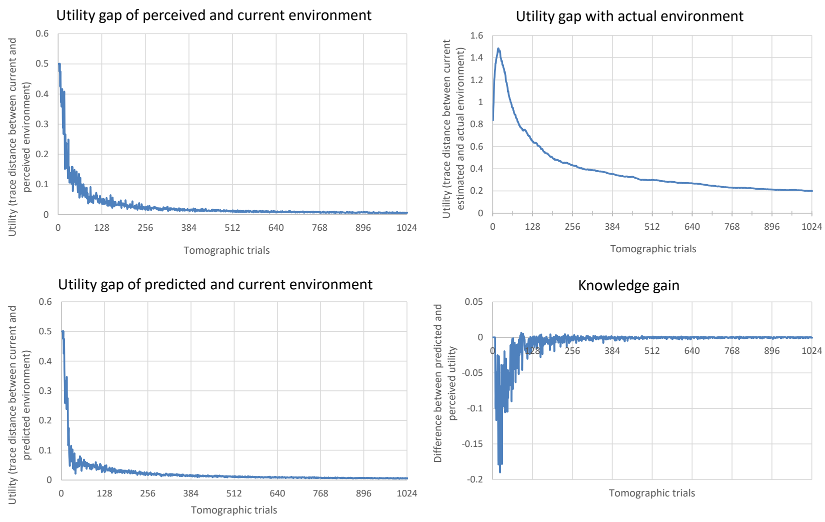


Figure 4.4: Trace distance EAQPT 1-qubit environment random unitary average of 20 experiments.

We defined a random 1-qubit unitary as the environment. Results of averaging over 20 random circuits are shown in Figure 4.4. The chosen distance measure chosen is trace distance. We found that Bures distance and Hilbert-Schmidt distance do not perform well. This is because the distance between the initial complete mixed state and the Choi matrix for the unitary in EAQPT is close to zero. The perceived and predicted utility are plotted on top-left and bottom-left, respectively. The perceived utility is the actual information gained by the agent on performing the action-perception interaction for the step. The predicted utility is the quantum generalization of the utility of KL-KSA. In this context, the quantum process represented as the Choi density matrix is the compressed representation of the environment. The difference between the predicted and perceived utility is the knowledge that reflect how well the current model agrees with the actual environment. In the top-right subplot, we show the remaining utility with the time step. It is only possible to know this when the target environment is known. While this is not the case for QKSA, we plot this to show the convergence of the learning behavior.

We show in the results that the perceived utility converges to zero. The striking advantage of this approach is that, without knowing the actual environment, this trend can be used to optimize a quantum algorithm. This can have a significant impact in algorithms like VQE, where quantum tomography is an integral part. Current tomography routines set a constant value of trial runs on the quantum computer, which is significantly more than the needed trials for typical statistical approximation threshold requirements. Via this online evaluation of the environment, it is possible to predict when the environment has been sufficiently modeled and abort the learning process. This can also be useful when the noise characteristic varies during the execution of the algorithm.

4.4. CONCLUSIONS AND OUTLOOK

In this work, we extend the formalism of universal artificial general intelligence (UAGI) to quantum environments. We generalized the KL-KSA to a quantum knowledge seeking agent (QKSA). The environment within the reinforcement learning setup is defined by an unknown quantum circuit in Qiskit. The agent models the environment using quantum process tomography algorithms. A quantum environment prevents the exact prediction of perceptions (as used by AIXI), as well as a single probability distribution of perception based on the set of actions (as used by KL-KSA). The probability distribution is conditioned on the chosen action and is thus represented by the more general density matrix formalism. Any quantum process can be represented as a Choi density matrix, which forms a model of the environmental dynamics.

Despite their theoretical significance, UAGI models are uncomputable, and thus are not useful for practical learning tasks. A typical solution is to restrict the runtime and length of the programs. Such solutions have been shown to learn simple games like Pac-Man. However, the space of programs grows exponentially, and thus a simple cutoff is not a scalable solution. To circumvent this, we propose to evaluate the algorithmic cost within a set of user provided Python codes instead of enumerating Turing machines. This considerably makes the framework more tractable.

Finally, the resource restrictions used in computable UAGI models (like UCAI and AIXI-tl) are arbitrary. In our model, these resource bounds are interdependent hyper-parameters whose value and trade-off relations are optimized using genetic programming. Thus, this allows open-ended evolution of the agents for changing environments. Each agent can self-replicate as a quine and thus is a recursive self-improving model of intelligence.

QKSA provides a framework to evaluate a swarm of UAGI agents that discover the resource tradeoffs in modeling a quantum environment. Besides the theoretical importance, the QKSA framework can be used to study the applicability of various distance measures of quantum information. It also has near term applicability in optimizing NISQ era variational quantum algorithms like QAOA, which rely on multiple runs of quantum tomography. We show as a proof-of-concept that it can be used in quantum process tomography where the QKSA knowledge gain reflects the trace distance with the unknown environment.

The entire OEE-URL framework of QKSA was highly interdisciplinary and theoretical compared to the previous chapters. Though a proof-of-concept using Qiskit was implemented and demonstrated, the eventual aim was to inform future directions of exploring the synergy between AIT and QC, specifically for causal modeling.

4.5. APPENDIX: ALGORITHMIC PROBABILITY AND QUINES

A dichotomy exist in algorithmic information theory, where, the length of the data string is considered a false indication of its complexity, while it does use the length of programs to calculate the likelihood of the occurrence of the data string. While short programs are considered more likely to produce a data string, the theory does not consider how likely it is for short programs to be generated by another higher level (meta-)program.

RESOURCE-BOUNDED ALGORITHMIC PROBABILITY

We consider a fixed length model (i.e. an universal linear bounded automata), where the data, the program, and all higher level programs have the same length, and are input programs or outputs of the same automata. This automata is denoted by $\mathcal{A}$. The length is a resource limitation from any realistic automata implementation. For naturally occurring computing hardware like DNA or the brain, it is imperative to have a fixed resource for storing the program, e.g. the number of base-pairs or the number of neurons. We explore the properties of the final distribution of the data string provided we consider multiple levels of meta-programs. The distribution converges to self-replicating programs or quines, which survive over generations of program-data hierarchy.

The universal Solomonoff algorithmic probability [108] of a program p on a (prefix-free) universal Turing machine (UTM) U for an output x is proportional to the sum of inverse of the description lengths of all programs that generate the output.

$$\xi_U(x) = \sum_{p:U(p)\to x} 2^{-l(p)}$$

This naturally formalizes Occam's razor (law of parsimony) and Epicurus's principle of multiple explanations by assigning larger prior credence to theories that require a shorter algorithmic description while considering all valid hypotheses.

Consider a Turing machine T with n symbols and m states, which can be enumerated by running a finite number of programs p. The algorithmic probability of a string x can be approximated as:

$$\xi(x) \approx D_{n,m}(x) = \frac{|T \in (n,m) : T(p) \text{ halts with output } x|}{|T \in (n,m) : T(p) \text{ halts}|}$$

i.e. counting how many programs produce the given output divided by the total number of programs that halt. The $|.|$ is used to denote set carnality throughout this section. This approximation is called the Coding Theorem Method (CTM) [113].

We are interested in the distribution of the computing output generated by a set of fixed size programs. We do not consider a special halt state, thus allowing us to explore the complete state space of programs [119] (i.e. the powerset of the full language of fixed length). This can encode programs that demonstrate a halting behavior (i.e. the tape output does not change after a certain time) by encoding the halting TM state as states of another TM that loop on themselves, moves the tape head arbitrarily and writes back the read character. For quantum processes and automata (that are generalizations of classical TM), the halting state cannot be defined explicitly. Only the output behavior can be observed on measurement at a user-defined time step since the start of the computation. Since we quantize the results of this formalism to a quantum case in QKSA, this consideration is crucial.

Let the description number of n symbols and m states be encoded as binary strings of length l. Thus, all 2^l possible programs have, and when run for t time steps (of course preferably larger than the Busy Beaver number), produces the following approximation of algorithmic probability:

$$D^{tl}_{n,m}(x) = \frac{|\mathcal{A}^t(p) \to x|}{2^l}$$

We reach the same result plugging in the constant size of programs in the original equation of $\xi(x)$. Note that, in the fixed length and time case, the automata is no longer guaranteed to be universal.

$$\xi(x)_{\mathcal{A}} \approx D^{tl}_{n,m}(x) = \sum_{p:\mathcal{A}(p)\to x} 2^{-l} = 2^{-l}|\mathcal{A}(p) \to x|$$

NESTING ALGORITHMIC PROBABILITY

Let's denote the resource-bounded algorithmic probability derived in the last section as:

$$\xi^{01}(x^0) = 2^{-l}|\mathcal{A}(x^1) \to x^0|$$

ξ^{01} denotes the algorithmic probability of the set of output strings x^0 and is based on considering a uniform distribution of the set of programs x^1. We drop the subscript $\mathcal{A}$ for brevity, but all results should be interpreted as having a dependence on the automata. This scalable notation considers fixed length programs and output data, no inputs and a specific automata $\mathcal{A}$. Thus, the set cardinalities are $|X^1| = |X^0| = 2^l$. Individual lengths of the strings, $x^1 \in X^1$ and $x^0 \in X^0$ are $l(x^1) = l(x^0) = l$.

In general, this is a many-to-one mapping, thus not all strings in X^0 are generated by running programs of the same size. The strings which are algorithmically random are not part of the set X^0 and are shown as the striped annulus in Figure 4.5.

The set of algorithmic probabilities ξ^{01} for all X is the resource-bounded approximation of the Solomonoff's universal (prior probability) distribution. The core motivation of algorithmic information theory to define the universal distribution from a uniform distribution is based on the idea that simpler shorter theories are more probable. Our fixed-length program (model/hypothesis) formulation seem to not allow shorter programs to have more weight. Yet, there will be more programs nevertheless to generate a simple data, e.g. 0101010101010101 can be either generated by looping 01 for 8 times, or 0101 for 4 times or 01010101 twice. Whichever is the most efficient of these 3 programs would print the output and reach a halting state early. What matters is the frequency of programs (as in the CTM), not at what stage in the computation of our t steps it reached a stable attractor state. So even within same length programs we expect a non-uniform distribution as we do for the data.

Now we can pose the question: 'why should all programs be considered equiprobably?' Considering a higher hierarchy of meta-programming, there is some physical pro-

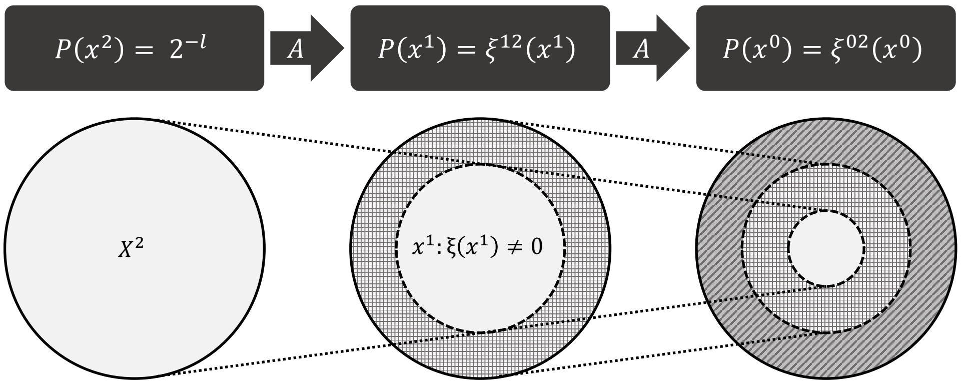


Figure 4.5: The space of programs typically map to a smaller set of output strings. The algorithm probability $P(x^0)$ of the output strings x^0 is based on a uniform probability 2^{-l} of the programs in the highest meta-level (left circle).

cess that generates that program on the program part of the tape of the automata. Normally, for the standard definition of algorithmic probability, it is considered an unbiased coin flip, or the infinite programming monkey theorem. Since the universal distribution is not known apriori, there is no other preference than a uniform distribution to start with.

Thus, the generalized form of the previous equation would be:

$$\xi^{02}(x^0) = 2^{-l} \sum_{x^1:\mathcal{A}(x^1)\to x^0} \left[\xi^{12}(x^1).2^{l}\right] = \sum_{x^1:\mathcal{A}(x^1)\to x^0} \xi^{12}(x^1)$$

If, $\xi^{12}(x^1) = 2^{-l}$ for all $x^1 \in X^1$, the definition would converge to the previous case, where the summation is simply counting the number of programs with the property $\mathcal{A}(x^1) \to x^0$.

However, we can feedback the universal distribution on the programs and understand what effect it has on the data. Thus introducing a hierarchy of automata levels, where the output data of one automata is the input program for the next levels. Weighting the contribution of each program based on the probability that they themselves physically occur on the program part of the tape.

Due to the invariance theorem, we can assume that the automata for the two levels are not same, with only constant overheads of translating (cross-compiling) the program of one machine to another, however, for our case, we use the same automata $\mathcal{A}$, i.e. $\xi^{01}_{\mathcal{A}_0}(x^0) = \xi^{01}_{\mathcal{A}}(x^0)$ and $\xi^{12}_{\mathcal{A}_1}(x^1) = \xi^{12}_{\mathcal{A}}(x^1)$.

We used the superscript 02 to distinguish it from the standard definition presented earlier (with the superscript 01). This denotes that, the final output is at level 0, while there are 2 levels of execution on the automata $\mathcal{A}$ that leads to this distribution, or 2 meta-levels. Note that now, $\xi^{23}(x^2) = 2^{-l}$, thus, while the programs (or, level 1 meta-programs) $x^1 \in X^1$ are no longer equiprobable, the level 2 meta-programs $x^2 \in X^2$ are drawn uniformly random.

Continuing this analogy, adding another meta-level would be of the form:

$$\xi^{03}(x^0) = \sum_{x^1:\mathcal{A}(x^1)\to x^0} \xi^{13}(x^1) = \sum_{x^1:\mathcal{A}(x^1)\to x^0} \left[\sum_{x^2:\mathcal{A}(x^2)\to x^1} \xi^{23}(x^2) \right]$$

Indeed it is possible to extend this argument to arbitrary many levels w, with no particular reason to choose ξ^{01} over ξ^{0w} for the expected distribution of the data occurring physically. This generalized recursive form is:

$$\xi^{ab}(x^a) = \sum_{x^{a+1}:\mathcal{A}(x^{a+1})\to x^a} \xi^{(a+1)b}(x^{a+1})$$

where, $\xi^{(w-1)w}(x^{w-1}) = 2^{-l}$

Let the number of strings $N^{0w} \leq 2^l$ with non-zero probability for a particular meta-level w be defined as:

$$N^{0w} = |\xi^{0w}(x^0) > 0| = \sum_{x^0 \in X^0} \lceil \xi^{0w}(x^0) \rceil$$

It can be easily seen that, since the programs are deterministic (i.e. has only 1 output), the program-data input-output map of the automata on the space of 2^l bit strings is a non-injective non-surjective function in the general case. Thus, for $r_1 > r_2$, $N^{0r_1} \leq N^{0r_2}$. We are interested in the properties as w approaches a large number.

- What are the properties of the strings that 'survive' over these generations (meta-levels)?
- Since each added hierarchy reduces the set size, at some threshold value of $r_1 > r_2 > w_t$ the inequality will become an equality $N^{0r_1} = N^{0r_2}$. Is $N^{0r_1} = N^{0r_2} \neq 0$ in that threshold?

FITNESS OF QUINES

A quine is a program which takes no input and produces a copy of its own source code as its output. It 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. The quine concept can be extended to multiple meta-levels, called ouroboros programs or quine-relays. Quines are also a limiting case of algorithmic randomness as their length is same as their output. If we consider the Turing machine automata, this translates to printing out the description on the tape which can be executed as a program by another Turing machine. Note that the rest of the Turing machine mechanism like the tape head and movement are akin to the underlying cellular automata rules that automatically apply to the new cells where the replicated machine manifests. The very design of the 3 parts of a constructor suggests that it cannot be algorithmically random as it should be possible to compress parts of its description. We are interested in the complexity and probability of constructors, which forms a subset of all possible configurations an automata can possess.

In our model, we consider the entire set of 2^l strings. Each string is represented by a node in Figure 4.6, with the arrows representing the mapping by running the string as a program on the automata. Thus, while many-to-one arrows are possible, one-to-many is not. We partition the set of strings (interpreted as program or data) into 2 subsets: attractors and repellers. Attractors are strings which when executed as a program generate as output a string which is also from the attractor subset. While, a repeller string when run as a program can generate either an attractor or a repeller string. The entire space might have multiple connected components. Each connected component consists of an attractor basin, made of quines or quine-relays, (as the multi-node attractor cycles) in Figure 4.6. Each node in the attractor basin might have a trail of repeller nodes, which, over cycles of algorithmic probability converge to the node on the basin.

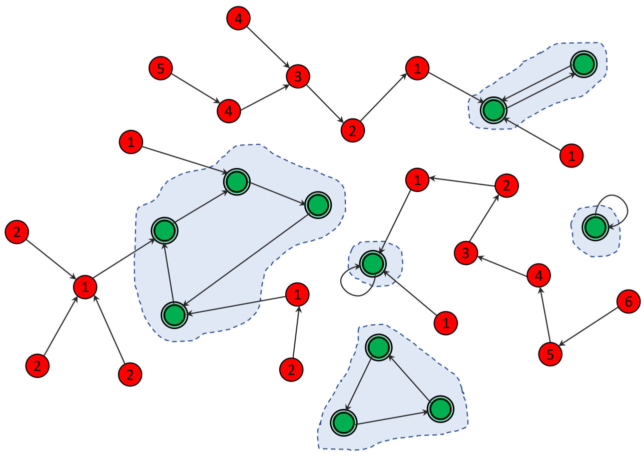


Figure 4.6: Attractor (green double-circled nodes) and repeller strings (red nodes).

Over multiple cycles the final set will only include the attractor nodes, with one-to-one mapping, thus conserving the number of strings in subsequent cycles. We denote this specific number of cycles with w_t, the meta-level at which a uniform distribution results in only attractors after w_t cycles. Each repeller node can be numbered as the number of hops away from an attractor basin. The highest hops is the w_t, i.e. 6 in the example in the figure. At this stage, each string has a one-to-one mapping. The number of attractors is given by

$$Q = N^{0w_t} = |\xi^{0w_t}(x^0) > 0|$$

The algorithmic probability for these constructor strings depends on the number of paths leading to these attractor basis over these cycles. w_t is dependent on the number of considered state and symbols, the specification of the automata, the length of the programs and the time approximation for estimating the algorithmic probability at each level. Being at least as (semi-) uncomputable as ξ, we can only study it under reasonable approximations via Experimental Algorithmic Information Theory (EAIT) [136, 204].

EXPERIMENTS

Here, we consider a particular case to illustrate the developed formalism of nesting algorithmic probability. We take the 2 state 2 symbol Linear Bounded Automata (LBA) as it is both non-trivial as well as within the bounds of exhaustive enumeration. The details of this machine can be found in [119]. The program (description number) is encoded as the list of transition functions for each state and read symbol:

$$[QMW]^{Q_1R_1}[QMW]^{Q_1R_0}[QMW]^{Q_0R_1}[QMW]^{Q_0R_0}$$

This gives the values $q_\delta = 12$, as the length of the description number required to store a program for this machine. Thus, the space of this encoding allows $P = 2^{12} = 4096$ possible programs. The tape is also of length $c = z = 12$ and consists of all zeros with the tape head on the left most character: $\underline{0}00000000000$ The machine is run for $z = q_\delta = 12$ iteration. A Python script that emulates this automata model for all 4096 cases is available at [205].

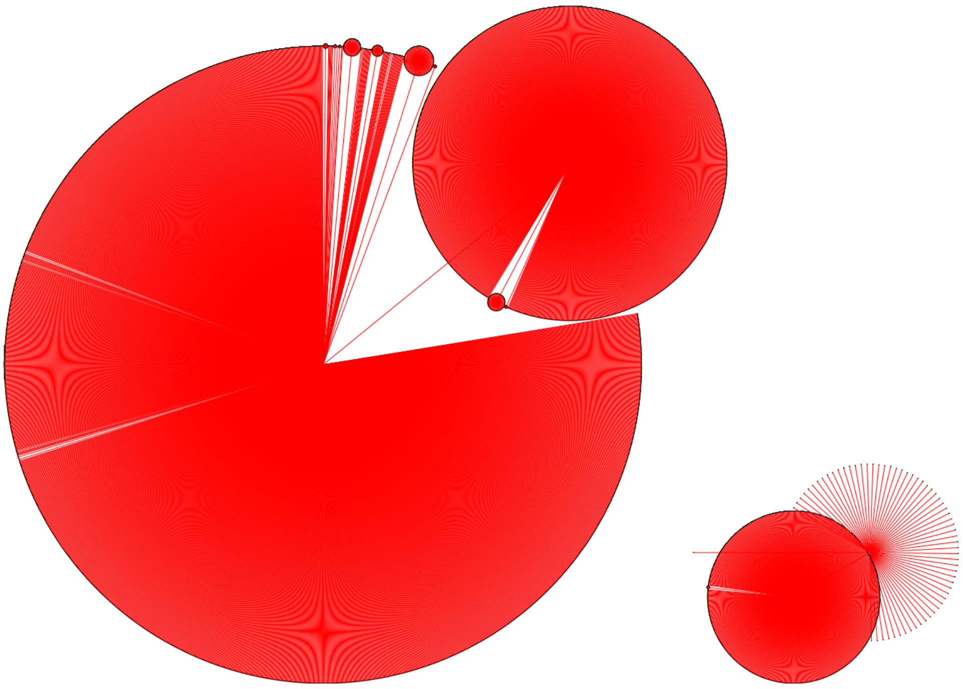


Figure 4.7: Level 3 of nesting algorithmic probability for a 2 state 2 symbol LBA (the high resolution svg is available at [205]).

At level 3 of nesting algorithm probability, we start with a uniform distribution of all programs, to produce the standard universal distribution. The mapping is shown in Figure 4.7 (the high resolution svg is available at [205]).

We observe that, at this level itself, the number of possible programs for the next generation gets reduced from 4096 to 21, with the following frequency of occurrence. The top 8 algorithmically probable attractor basins are shown in Figure 4.8.

```
P0    : 1886     P2048 : 1147     P4095 : 640      P3072 : 110
P1365 : 64       P2047 : 64       P2730 : 64       P1024 : 41
P3840 : 17       P128  : 11       P3968 : 11       P1344 : 10
P3584 : 10       P4032 : 10       P1792 : 2        P1920 : 2
P2560 : 2        P2688 : 2        P192  : 1        P1728 : 1
P2944 : 1
```

Also, at this level itself we find that the machines P0 and P4095 are the only self-replicating programs, thus the limiting behavior can already be predicted.

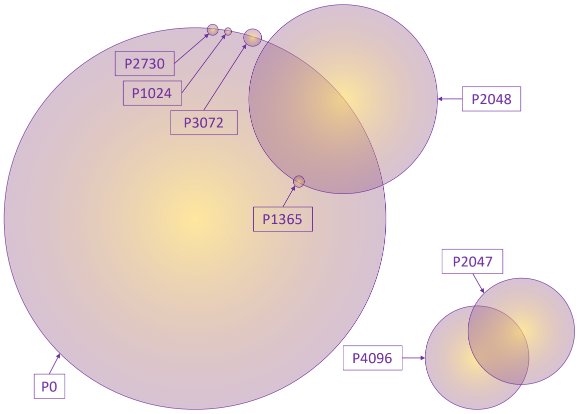


Figure 4.8: The 8 largest attractor basin of level 3 of nesting algorithmic probability.

At level 2, these 21 programs get further mapped to just 3 programs, with the following frequency.

```
P0    : 16       P4095 : 3        P2048 : 2
```

At level 1, the 3 programs finally converge to the quines P0 and P4095.
The frequency is as follows:

```
P0    : 2        P4095 : 1
```

At level 0, we reach the attractor states with a uniform one-to-one mapping.
Level 2, 1 and 0 are shown in Figure 4.9,4.10 and 4.11 respectively.
Now, to calculate the overall algorithmic probability of these two fixed points, we calculate the cumulative frequency over these 4 levels. Thus, at level 2, the total frequency of P4095 consists of adding up the frequency of programs P4095, P2047, P4032 from the previous step, totaling to $640 + 64 + 10 = 714$. For P2048, we add up the frequency of Programs P1365, P1344, totaling to $64 + 10 = 74$. The rest of the 16 programs total to 3308 cases that reach P0. At level 1, P2048 also reaches the P0 attractor, giving the final algorithmic frequency of the quines as:

```
P0    : 3382     P4095 : 714
```

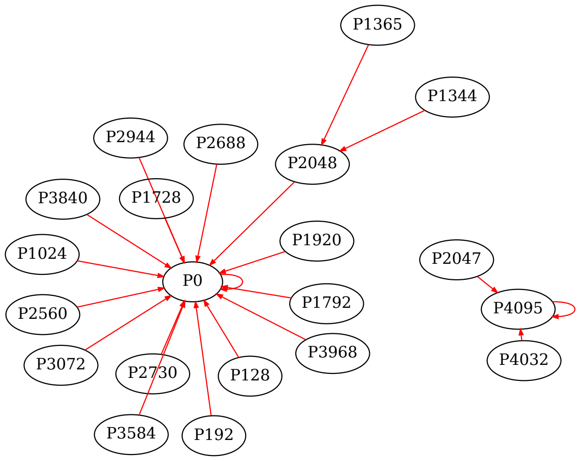


Figure 4.9: Level 2 of nesting algorithmic probability for a 2 state 2 symbol LBA.

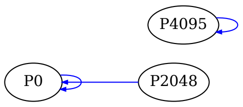


Figure 4.10: Level 1 of nesting algorithmic probability for a 2 state 2 symbol LBA.

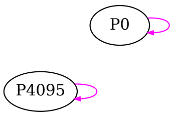


Figure 4.11: Level 0 of nesting algorithmic probability for a 2 state 2 symbol LBA.

We call this the resource-bounded Universal Constructor distribution.

A similar experimentation on this space of 4096 programs is conducted as part of the Wolfram Physics Project exploring the rulial space of Turing machines [206]. The 2-2-1 LBA with 4096 programs does not show much diversity resulting in simple quines. It remains to be seen what the encodings of quines in larger spaces reveal about the structure and complexity of constructors.

5 Conclusion

The goal of this doctoral research was to **identify high-impact long-term applications of quantum computation and formulate corresponding quantum algorithms to compute them**. This forms the algorithm layer of the quantum accelerator stack. In this dissertation, three specific case studies are presented.

§ 2 studies quantum algorithms to accelerate DNA sequence reconstruction. This is a crucial step in the genomics data processing pipeline. Two different ways of reconstruction were targeted, and the corresponding quantum computing approaches were developed.

In § 3 the quantum versions of classical automata like Turing machines and linear bounded automata were implemented. This corresponds to executing a superposition of all classical programs of a particular size with a specified encoding. We show that such a quantum oracle can be used for approximating mathematical objects in algorithmic information theory. Understanding these has various applications in causal modeling use cases like genome analysis.

In § 4 we study reinforcement learning agents based on universal automata. These models are generalized for learning in a quantum environment, as well as, an open-ended evolution of the utility based on the resources used in the modeling and prediction process. The proposed agent is a general intelligence formalism that reflects the mechanism behind the human understanding of quantum mechanics. This can be used to automate the design and optimization of quantum experiments.

5.1. Scientific contributions

As a summary, the major scientific contributions of this doctoral research for each use case are:

1. **Quantum-accelerated genome sequence reconstruction**
    (a) first quantum algorithm to accelerate approximate index search that preserves the quadratic speedup of Grover search
    (b) implementation of DNA sequence reconstruction based on reference alignment on a quantum computing simulator

   (c) implementation of de novo read alignment for DNA sequence reconstruction using the quantum approximate optimization algorithm
   (d) implementation of de novo read alignment for DNA sequence reconstruction using a QUBO formulation on a quantum annealing hardware
   (e) demonstration of proof-of-concept for QiBAM (8 qubits, 16-base reference, 3-base read) on QX and OpenQL, and QuASeR (16 qubits, 4 reads) on QX and OpenQL and (60 qubits, 4 reads) on Ocean and D-Wave 2000Q
2. **Quantum automata for algorithmic information**
   (a) first gate-level design of a quantum universal linear bounded automata
   (b) an application framework for specific algorithmic information mathematical objects using a quantum superposition of classical automata
   (c) implementation and demonstration of proof-of-concept for 1-state 2-symbol 1-dimension QPULBA (16 qubits) for quines and a unit level simulation for 2-state 2-symbol 1-dimension QPULBA on QX, OpenQL, and Qiskit
3. **Universal reinforcement learning in quantum environments**
   (a) a mathematical description and enumeration of nested algorithmic probability for program-data duality showing the convergence to constructors
   (b) a formulation of a universal reinforcement learning agent with an evolving utility function based on computational resources that undergoes recursive self-improvement
   (c) first universal reinforcement learning agent model for quantum environments
   (d) implementation and demonstration of proof-of-concept QKSA framework for 2 EAQPT strategies using trace distance on 1 qubit environment on Qiskit

5.2. MAIN INSIGHTS

Beyond the three specific use cases, this research provides general insights into the applications of quantum computation. These are summarized in this section.

Interdisciplinary knowledge and research collaborations —

Quantum algorithm design is in many ways orthogonal to classical algorithm design. While knowledge of classical algorithms and their associated theoretical complexity and practical application is imperative, it helps very little in developing new quantum algorithms. A good grasp of quantum information theory (and associated subjects like complex analysis, linear algebra, quantum mechanics, algorithms, and digital logic) is necessary to design these approaches. Due to this, quantum algorithm design needs an interdisciplinary collaborative research team. Various focused tracks and graduate programs that are being initiated by academic institutions would eventually help to upskill the future quantum computing workforce.

Balancing requirements from quantum hardware and application —

Quantum algorithm design is transitioning from theoretical description to an implementation focus. This has both advantages and disadvantages. It helps to understand the full application pipeline and the implementation details of the specific algorithm in question. This clarifies the stages like data input, oracle calls, and measurement strategies that are typically abstracted in the mathematical descriptions. However, the surge to experimentally demonstrate these application pipelines on currently available NISQ processors to attract research funding is often scientifically futile. These results unequivocally demonstrate that a quantum advantage for the target applications is beyond the reach of the state-of-the-art QPUs. Better quantum computing benchmarks [207] with an application focus can eliminate the need for this exercise by informing the algorithm designers of the capability of the quantum hardware to prove advantageous in an experimental setting. In our research, we found balancing between these two to be an optimal strategy, where the algorithmic description of the application pipeline is designed as a quantum program to understand the gate and qubit complexity of the full quantum logic. This can inform the quality and size of QPUs required to demonstrate application-motivated problem sizes.

Gate-level logic design for novel quantum algorithms —

The level of quantum algorithm programming is still significantly reliant on logic gate level design. This level is on par with hardware-description language (like VHDL and Verilog) and assembly languages (like x86 assembly). Most popular quantum programming languages are implemented as libraries over classical high-level languages (like C++ or Python) where the programming constructs like decisions and loops use the native language, however, the quantum kernel still needs to be described by appending individual quantum logic gates onto an object or data structure. Many application-focused libraries (e.g., for molecular simulation or combinatorial optimization) enable application developers to use quantum acceleration in a well-abstracted framework. However, for the foreseeable future, designing novel quantum algorithms will require reasoning at the gate level.

High-level quantum programming languages and compiler-level automation —

Vertical integration of a quantum algorithm on the computing stack involves other optimizations and gate operations. These include, for example, quantum error correction, mapping, and routing on the QPU topology, compilation to the native gate-set of the target QPU, etc. In the NISQ era, these are often discussed as part of the quantum algorithm as a hardware-software co-design strategy. Looking forward, these auxiliary steps will eventually be entirely automated by the lower layers of program optimization by the compiler and micro-architecture. Since the proof of concepts of quantum algorithms in the NISQ era are widely removed from the problem size where it would achieve quantum advantage, we promote encapsulating these features as part of the quantum accelerator that would be considered for benchmarking. The quantum algorithm designer can then focus on the logical transformations that need to be programmed for the application use case. We call this approach the perfect intermediate-scale quantum (PISQ) computing.

'Small data, big compute' as the target for quantum-accelerated application — Lastly, we found that the features of quantum computation provide a computational benefit when the algorithm can harness a specific form. Each accelerator has a similar form, e.g., for GPUs, this form is called embarrassing parallelism. For QPUs, the form is 'small data big compute.' Thus, quantum algorithms would unlikely be the solution for big data problems and data-driven machine learning. The exponentially large state space in quantum should be harnessed without negating the promising speedup in the data encoding and measurement complexity. Restricting the exploration to the sub-space where no efficient classical algorithms and heuristics exist is beneficial till large-scale universal QCs are built. However, these quantum algorithms might find solutions to the computational bottleneck by using very different kernels. Such kernels might be more suited for quantum than classical computation and thus typically not popular in current algorithm research. This research focused on such promising quantum algorithms by exploring quantum formulations that use causal models, symbolic manipulations, and algorithmic information.

5.3. FUTURE DIRECTIONS

As with most doctoral research, the various research decisions were undertaken based on what was perceived as the most promising direction at that moment. Nonetheless, each use case suggests future directions which can be pursued further. Some of these are listed here.

- **Demonstration on a quantum processor** — The overarching goal of all quantum circuits and algorithms is to execute and demonstrate the application on a quantum processor. This was only possible for the de novo read assembly, where the simulated results were verified on the D-Wave quantum annealer. The other algorithms, being gate-based, are currently beyond [208] the capability of available quantum processors. While all the algorithms were verified using PISQ simulation, the real speedup cannot be demonstrated without executing on a quantum processor.
- **Grover's adaptive search for optimization** — Techniques based on Grover search (like QiBAM) have a provable speedup, while those based on variational hybrid heuristic optimization (like QuASeR) are more promising in the NISQ era. Some recent techniques represent a trade off between these approaches. Grover's adaptive search [209] and accelerated variational quantum eigensolver [210] are two such examples that trade off between the near-term and fault-tolerant regimes of quantum algorithm design for optimization and simulation problems, respectively.
- **Quantum accelerated causal inference** — We developed the primitives for exploring the space of programs on an automata model. Unlike quantum/classical data-driven machine learning, this preserves the explanatory power of the model the learning converges to. This can be used for machine learning tasks [211]. Specifically to bioinformatics, this can be applied for quantum accelerated causal hypothesis testing [212] on gene regulatory networks.
- **Multi-agent modeling of quantum environment** — Quantum mechanics is inherently a two-party relational theory. Thus it is interesting to understand how multiple classical agents can agree on the dynamics of a quantum environment while having different action-perception capabilities. This is studied within quantum communication protocols. Specifically, coarse-grained quantum descriptions [213] in multiple observers is a viable extension of the QKSA framework when the pool of QKSA agents that are part of the population can interact. In a similar fashion, crossover operators can also be added to the mutating reinforcement learning agents.
- **Uncomplexity as a computational resource** — The QKSA framework proposes five computational resources to define the utility function for the agent. While the

program length, memory, time, and approximation have been studied in many contexts, energy (or the thermodynamic cost of algorithms) is much less developed theoretically. Practically these are assessed by the power consumption on a specific computer. Very recently, this is being studied for classical Turing machines [203], but the work is still at an early stage to predict the energy cost of a high-level language program. The uncomplexity metric [214] ties together these resources in the many-body physics and cosmology setting. Understanding the impact of uncomplexity for a generic quantum algorithm would be beneficial not only for understanding the limits of quantum speedup [202] but also for the convergence of the QKSA to a certain form of the utility function. In the agent-environment scenario, generalizing this to 'relative uncomplexity' might more formally explain scientific model discovery in the light of no free lunch theorems [215].

- **Entanglement as a computational resource —** Entanglement (between the agent and the environment) is also another interesting computational resource that we did not consider due to our assumption that the agent, like human reasoning, is classical. Thus, a future generalization could involve studying QKSA with quantum communication and computing capabilities. The entanglement can be used as a world-making relation [216] instead of space-time locality. The agent in such a case should be modeled as an embedded AI [217], governed by the same laws of physics as that of the environment. An embedded QKSA can be studied in a unified toy universe like AdS-CFT [218] using tensor networks [219].

The contributions of this research bring into the spotlight various synergies among the fields of 'quantum information and computation', 'algorithmic information theory', 'genomics' and 'reinforcement learning'. Further research on these interdisciplinary links would prove beneficial in many other scientific and societal applications.

Epilogue

My interactions with the world of knowledge - the inputs of education and the outputs of research have already been presented. Here, I intend to highlight aspects of the program, that internally weaves together these threads.

Prior to my research career in quantum computing, I was a scientist at Indian Space Research Organisation (ISRO), the national space agency of India. This childhood dream of being a space scientist was fueled by the curiosity that the mysterious night sky infused in me. It translated into me choosing an undergraduate program at the Indian Institute of Space Science and Technology, focused on aviation electronics, computer science and space robotics. Thereafter, at ISRO, I was the lead software architect for mission-critical firmware on board 10 successful remote-sensing satellite missions. These encompass cartography, resource mapping, space observatory, and lunar missions. It was overwhelmingly gratifying to play an active role in these engineering frontiers, and partake in the joy of my codes helping in the discovery of one of the earliest galaxies (via AstroSat), and affirm the detection of water while orbiting the moon (via Chandrayaan-2).

In days where computers are being strapped to spectacles and voyages of landing on asteroids making news, I realized both had their limitations. Space exploration is strangled by the speed of spacecrafts and more fundamentally by light speed. Computer technology is limited by transistor miniaturization limits and computational complexity alike. My search for a concrete inkling, to relinquish my job for an academic research career, converged to quantum computing (QC). I realized, to make concrete progress in QC, an interdisciplinary perspective is indispensable. Fortunately, over the last century the boundaries between the various branches of science is noticeably fading. These advances allow applying theories and techniques from one discipline to reinforce conjectures in another. It allows exploring fundamental ideas (like emergence and universality) at the intersection of mathematics, computer science, physics and biology. My doctoral research is a small step towards this larger quest.

Computer science developed into a fundamental discipline, when the incompleteness of logical systems was complemented with the definition of universal computation. While its link with physics via information theory and thermodynamics were already proposed, this was concretely established with the formalization of a quantum mechanical computational automata model. Two catch-phrases summarize this connection: ‘it from bit’ - meaning that everything in the universe can be modeled as a digital computation, and, ‘bit from it’ - meaning every computational process is physically embodied, thus respects physical laws. Digital physics, as this field is called, was my main impetus to study quantum computing. Research on similar ideas are now increasingly being pursued. Some such examples include, generalized holographic principle, tensor networks, informational axioms of quantum mechanics, principle of computational equivalence, a code-theoretic approach to physics, causal sets, quantum cellular automata, and multiway computation.

Another intriguing notion that has always charmed me is the principle of emergence. Besides the possibility that the universe itself is emergent, two more accessible, yet equally obscure examples are, life and intelligence. Biochemical processes like metabolism and reproduction, orchestrates the spectacle of life via inanimate molecules encoding the genome. This can be studied purely as a digital computation unfolding within the cells. It is a program, which depending on environmental inputs act out these processes - eventually evolving the next generation with a compressed memory of the most eventful of these stimuli by natural selection. This notion was the major drive behind half of the use cases presented in this dissertation. Connecting this view of biology to computer science and physics has led to various seminal concepts like, evolutionary algorithms, meta-biology, cellular automata, self-replication, free energy principle, artificial life and constructor theory.

Artificial intelligence (AI) is the third aspect that influenced my research. The triumph of data-driven machine learning increasingly impact aspects of our daily life. Automating more critical applications needs trustable decisions based on causal insight, what is pursued as understandable/explainable/interpretable AI. The confidence in the current capabilities now also permits technical discussions of the original motivation of AI, general intelligence. I explored how quantum computational benefits can be applied to models like neuro-evolution, AutoML, BDM and AIXI. This led to an in-depth exploration of practical applications of algorithmic information, in QC, genomics and AI. Given the powers of computational universality and algorithmic information, I wanted to understand how general intelligence can encompass the counter-intuitive aspects of quantum information, like entanglement and measurement. The QKSA framework elegantly melds various concepts like, genetic programming, self-replication (quines) and constructors, reinforcement learning, swarm intelligence and emergence, program-data duality, computational universality, and algorithmic probability.

When I started this doctoral research, I championed the 'pancomputationalist' ideology, i.e., the evolving cosmos is best expressed as a coherence of toggling (qu)bits, on a gigantic (quantum) computer; a simulation unfolding itself. By the end of my doctoral research, I identify myself as an 'algorithmic absurdist'. **Algorithmic absurdism** maintains that the universe as a whole has no information in accordance to the no free lunch theorems, though that ontic view is not physically observable and thus this unification would remain inaccessible and unprovable. Splitting the universe into an agent and an environment however permits a general technique for building relational epistemic models. This optimal inductive modeling technique by computationally bounded participators is based on resource-aware algorithmic information that adjusts its utility via open-ended evolution. I believe this will be the guiding philosophy of science in the days to come.

BIBLIOGRAPHY

REFERENCES

[1] Karl Rupp. 42 years of microprocessor trend data | karl rupp. Available at https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/, 2018.

[2] Igor L Markov. Limits on fundamental limits to computation. *Nature*, 512(7513):147, 2014.

[3] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K. Bertels, G.K. Kuzmanov, and E. Moscu Panainte. The molen polymorphic processor. *IEEE Transactions on Computers*, 53:1363–1375, 2004.

[4] Leon Chua. Resistance switching memories are memristors. In *Memristor Networks*, pages 21–51. Springer, 2014.

[5] Leonard M Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–1024, 1994.

[6] Koen Bertels, A Sarkar, T Hubregtsen, M Serrao, AA Mouedenne, A Yadav, A Krol, and I Ashraf. Quantum computer architecture: Towards full-stack quantum accelerators. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2020.

[7] Richard P Feynman. There's plenty of room at the bottom: An invitation to enter a new field of physics. In *Handbook of Nanoscience, Engineering, and Technology, Third Edition*, pages 26–35. CRC Press, 2012.

[8] Charles H Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM journal on Computing*, 26(5):1510–1523, 1997.

[9] Scott Aaronson. The limits of quantum computers. *Scientific American*, 298(3):62–69, 2008.

[10] Gordon E Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, 1998.

[11] Zsolt Tőkei. Logic scaling options for the next 10 years: From finfet to cfet, from dual damascene to semi damascene. In *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 49–49, 2022.

[12] David Deutsch. Quantum theory, the church-turing principle and the universal quantum computer. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 400, pages 97–117. The Royal Society, 1985.

[13] David Elieser Deutsch. Quantum computational networks. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 425(1868):73–90, 1989.

[14] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum computation by adiabatic evolution. *arXiv preprint quant-ph/0001106*, 2000.

[15] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

[16] Christoph Adami. On the origin of quantum uncertainty. *arXiv preprint arXiv:2005.07325*, 2020.

[17] David P DiVincenzo et al. The physical implementation of quantum computation. *arXiv preprint quant-ph/0002077*, 2000.

[18] L Riesebos, X Fu, AA Moueddenne, L Lao, S Varsamopoulos, I Ashraf, J van Someren, N Khammassi, CG Almudever, and K Bertels. Quantum accelerated computer architectures. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 2019.

[19] X Fu, MA Rol, CC Bultink, J van Someren, N Khammassi, I Ashraf, RFL Vermeulen, JC De Sterke, WJ Vlothuizen, RN Schouten, et al. An experimental microarchitecture for a superconducting quantum processor. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 813–825. ACM, 2017.

[20] Lingling Lao, Daniel M Manzano, Hans van Someren, Imran Ashraf, and Carmen G Almudever. Mapping of quantum circuits onto nisq superconducting processors. *arXiv preprint arXiv:1908.04226*, 2019.

[21] Savvas Varsamopoulos, Koen Bertels, and Carmen G Almudever. Decoding surface code with a distributed neural network–based decoder. *Quantum Machine Intelligence*, 2(1):1–12, 2020.

[22] Nader Khammassi, Gian G Guerreschi, Imran Ashraf, Justin W Hogaboam, Carmen G Almudever, and Koen Bertels. cqasm v1.0: Towards a common quantum assembly language. *arXiv preprint arXiv:1805.09607*, 2018.

[23] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[24] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[25] Aritra Sarkar, Zaid Al-Ars, and Koen Bertels. Estimating algorithmic information using quantum computing for genomics applications. *Preprints*, 2021.

[26] Zachary D Stephens, Skylar Y Lee, Faraz Faghri, Roy H Campbell, Chengxiang Zhai, Miles J Efron, Ravishankar Iyer, Michael C Schatz, Saurabh Sinha, and Gene E Robinson. Big data: astronomical or genomical? *PLoS biology*, 13(7):e1002195, 2015.

[27] Margaret A Hamburg and Francis S Collins. The path to personalized medicine. *New England Journal of Medicine*, 363(4):301–304, 2010.

[28] Kris A Wetterstrand. The cost of sequencing a human genome. Available at `https://www.genome.gov/27565109/`, 2018.

[29] Aritra Sarkar. Quantum algorithms for pattern-matching in genomic sequences. Master's thesis, Delft University of Technology, 6 2018.

[30] Broad Institute. Broad institute gatk best practices pipeline. Available at https://www.broadinstitute.org/partnerships/education/broade/best-practices-variant-calling-gatk-1, 2018.

[31] Ernst Joachim Houtgast, Vlad-Mihai Sima, Koen Bertels, and Zaid Al-Ars. Hardware acceleration of bwa-mem genomic short read mapping for longer read lengths. *Computational biology and chemistry*, 75:54–64, 2018.

[32] Dan Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge university press, 1997.

[33] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

[34] Lov K Grover. Quantum mechanics helps in searching for a needle in a haystack. *Physical review letters*, 79(2):325, 1997.

[35] Christof Zalka. Grover's quantum searching algorithm is optimal. *Physical Review A*, 60(4):2746, 1999.

[36] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik: Progress of Physics*, 46(4-5):493–505, 1998.

[37] Eli Biham, Ofer Biham, David Biron, Markus Grassl, and Daniel A Lidar. Grover's quantum search algorithm for an arbitrary initial amplitude distribution. *Physical Review A*, 60(4):2742, 1999.

[38] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum counting. In *International Colloquium on Automata, Languages, and Programming*, pages 820–831. Springer, 1998.

[39] MP John. Sampling with quantum mechanics. *arXiv preprint quant-ph/0306181*, 2003.

[40] George F Viamontes, Igor L Markov, and John P Hayes. Is quantum search practical? *Computing in science & engineering*, 7(3):62–70, 2005.

[41] P Mateus and Y Omar. Quantum pattern matching. *arXiv preprint quant-ph/0508237*, 2005.

[42] Dan Ventura and Tony Martinez. Quantum associative memory with exponential capacity. In *Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on*, volume 1, pages 509–513. IEEE, 1998.

[43] Dan Ventura. Artificial associative memory using quantum processes. In *Proceedings of the International Conference on Computational Intelligence and Neuroscience*, volume 2, pages 218–221, 1998.

[44] Dan Ventura and Tony Martinez. Initializing the amplitude distribution of a quantum state. *Foundations of Physics Letters*, 12(6):547–559, 1999.

[45] Dan Ventura and Tony Martinez. A quantum associative memory based on grover's algorithm. In *Artificial Neural Nets and Genetic Algorithms*, pages 22–27. Springer, 1999.

[46] Dan Ventura and Tony Martinez. Quantum associative memory. *Information Sciences*, 124(1-4):273–296, 2000.

[47] AA Ezhov, AV Nifanova, and Dan Ventura. Quantum associative memory with distributed queries. *Information Sciences*, 128(3-4):271–293, 2000.

[48] J-P Tchapet Njafa, SG Nana Engo, and Paul Woafo. Quantum associative memory with improved distributed queries. *International Journal of Theoretical Physics*, 52(6):1787–1801, 2013.

[49] Lloyd CL Hollenberg. Fast quantum search algorithms in protein sequence comparisons: Quantum bioinformatics. *Physical Review E*, 62(5):7532, 2000.

[50] K. Giannakis, C. Papalitsas, G. Theocharopoulou, S. Fanarioti, and T. Andronikos. A quantum-inspired optimization heuristic for the multiple sequence alignment problem in bio-computing. In *2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA)*, pages 1–8, 2019.

[51] Koen Bertels, Aritra Sarkar, and Imran Ashraf. Quantum computing—from nisq to pisq. *IEEE Micro*, 41(5):24–32, 2021.

[52] Craig Gidney. Constructing large controlled nots. Available at `http://algassert.com/circuits/2015/06/05/Constructing-Large-Controlled-Nots.html`, 2018.

[53] Michel Eduardo Beleza Yamagishi. *Mathematical Grammar of Biology*. Springer, 2017.

[54] Vivek V Shende, Stephen S Bullock, and Igor L Markov. Synthesis of quantum-logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(6):1000–1010, 2006.

[55] Anna M. Krol, Aritra Sarkar, Imran Ashraf, Zaid Al-Ars, and Koen Bertels. Efficient decomposition of unitary matrices in quantum circuit compilers. *Applied Sciences*, 12(2), 2022.

[56] Nader Khammassi, Imran Ashraf, JV Someren, Razvan Nane, AM Krol, M Adriaan Rol, Lingling Lao, Koen Bertels, and Carmen G Almudever. Openql: A portable quantum programming framework for quantum accelerators. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 18(1):1–24, 2021.

[57] Nader Khammassi, I Ashraf, X Fu, Carmen G Almudever, and Koen Bertels. Qx: A high-performance quantum computer simulation platform. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 464–469. IEEE, 2017.

[58] Aritra Sarkar, Zaid Al-Ars, and Koen Bertels. Quaser–quantum accelerated de novo dna sequence reconstruction. *arXiv preprint arXiv:2004.05078*, 2020.

[59] AS Boev, AS Rakitko, S Usmanov, AN Kobzeva, IV Popov, VV Ilinsky, EO Kiktenko, and AK Fedorov. Genome assembly using quantum and quantum-inspired annealing. *arXiv preprint arXiv:2004.06719*, 2020.

[60] Abdul Rafay Khan, Muhammad Tariq Pervez, Masroor Ellahi Babar, Nasir Naveed, and Muhammad Shoaib. A comprehensive study of de novo genome assemblers: current challenges and future prospective. *Evolutionary Bioinformatics*, 14:1176934318758650, 2018.

[61] Jason R Miller, Sergey Koren, and Granger Sutton. Assembly algorithms for next-generation sequencing data. *Genomics*, 95(6):315–327, 2010.

[62] Jennifer Commins, Christina Toft, and Mario A Fares. Computational biology methods and their application to the comparative genomics of endocellular symbiotic bacteria of insects. *Biological procedures online*, 11(1):52, 2009.

[63] Gary Kochenberger, Jin-Kao Hao, Fred Glover, Mark Lewis, Zhipeng Lü, Haibo Wang, and Yang Wang. The unconstrained binary quadratic programming problem: a survey. *Journal of Combinatorial Optimization*, 28(1):58–81, 2014.

[64] Fred Glover and Gary Kochenberger. A tutorial on formulating qubo models. *arXiv preprint arXiv:1811.11538*, 2018.

[65] Philipp Hauke, Helmut G Katzgraber, Wolfgang Lechner, Hidetoshi Nishimori, and William D Oliver. Perspectives of quantum annealing: Methods and implementations. *Reports on Progress in Physics*, 83(5):054401, 2020.

[66] Yudong Cao, Jonathan Romero, Jonathan P Olson, Matthias Degroote, Peter D Johnson, Mária Kieferová, Ian D Kivlichan, Tim Menke, Borja Peropadre, Nicolas PD Sawaya, et al. Quantum chemistry in the age of quantum computing. *arXiv preprint arXiv:1812.09976*, 2018.

[67] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.

[68] Stuart Hadfield, Zhihui Wang, Bryan O'Gorman, Eleanor G Rieffel, Davide Venturelli, and Rupak Biswas. From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. *Algorithms*, 12(2):34, 2019.

[69] Jarrod R McClean, Sergio Boixo, Vadim N Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature communications*, 9(1):1–6, 2018.

[70] Giacomo Nannicini. Performance of hybrid quantum-classical variational heuristics for combinatorial optimization. *Physical Review E*, 99(1):013304, 2019.

[71] William J Cook. *In pursuit of the traveling salesman: mathematics at the limits of computation*. Princeton University Press, 2011.

[72] Koen Mesman, Zaid Al-Ars, and Matthias Möller. Qpack: Quantum approximate optimization algorithms as universal benchmark for quantum computers. *arXiv preprint arXiv:2103.17193*, 2021.

[73] Gian Giacomo Guerreschi and Mikhail Smelyanskiy. Practical optimization for hybrid quantum-classical algorithms. *arXiv preprint arXiv:1701.01450*, 2017.

[74] Sami Khairy, Ruslan Shaydulin, Lukasz Cincio, Yuri Alexeev, and Prasanna Balaprakash. Learning to optimize variational quantum circuits to solve combinatorial problems. *arXiv preprint arXiv:1911.11071*, 2019.

[75] Ruslan Shaydulin, Ilya Safro, and Jeffrey Larson. Multistart methods for quantum approximate optimization. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–8. IEEE, 2019.

[76] Hector Zenil. A review of methods for estimating algorithmic complexity: Options, challenges, and new directions. *Entropy*, 22(6):612, 2020.

[77] Fernando Soler-Toscano, Hector Zenil, Jean-Paul Delahaye, and Nicolas Gauvrit. Calculating kolmogorov complexity from the output frequency distributions of small turing machines. *PloS one*, 9(5), 2014.

[78] Gregory Chaitin. *Proving Darwin: making biology mathematical*. Vintage, 2012.

[79] Sydney Brenner. Life's code script. *Nature*, 482(7386):461–461, 2012.

[80] Abel Molina and John Watrous. Revisiting the simulation of quantum turing machines by quantum circuits. *Proceedings of the Royal Society A*, 475(2226):20180767, 2019.

[81] David B Searls. The language of genes. *Nature*, 420(6912):211–217, 2002.

[82] Marc D Hauser and Jeffrey Watumull. The universal generative faculty: The source of our expressive power in language, mathematics, morality, and music. *Journal of Neurolinguistics*, 43:78–94, 2017.

[83] Stephen Wolfram. *A new kind of science*, volume 5. Wolfram media Champaign, IL, 2002.

[84] Jack C Chaplin, Natalio Krasnogor, and Noah A Russell. Photochromic molecular implementations of universal computation. *Biosystems*, 126:12–26, 2014.

[85] Scott Aaronson. Complexity zoo. Available at `https://complexityzoo.net/Complexity_Zoo`, 2021.

[86] Neil Immerman. *Descriptive complexity*. Springer Science & Business Media, 2012.

[87] Gabriel Robins. The extended chomsky hierarchy reloaded. Available at `http://www.cs.virginia.edu/~robins/cs6160/slides/Theory_Lecture_17-20.pdf`, 2022.

[88] Scott Aaronson. $P \overset{?}{=} NP$. In *Open problems in mathematics*, pages 1–122. Springer, 2016.

[89] Paul Benioff. The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. *Journal of statistical physics*, 22(5):563–591, 1980.

[90] Paul Benioff. Quantum mechanical hamiltonian models of turing machines. *Journal of Statistical Physics*, 29(3):515–546, 1982.

[91] A Chi-Chih Yao. Quantum circuit complexity. In *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*, pages 352–361. IEEE, 1993.

[92] Satoshi Iriyama, Masanori Ohya, and Igor Volovich. Generalized quantum turing machine and its application to the sat chaos algorithm. In *Quantum Information and Computing*, pages 204–225. World Scientific, 2006.

[93] Simon Perdrix and Philippe Jorrand. Classically controlled quantum computation. *Mathematical Structures in Computer Science*, 16(4):601–620, 2006.

[94] Qisheng Wang and Mingsheng Ying. Quantum random access stored-program machines. *arXiv preprint arXiv:2003.03514*, 2020.

[95] Mingsheng Ying, Li Zhou, and Yangjia Li. Reasoning about parallel quantum programs. *arXiv preprint arXiv:1810.11334*, 2018.

[96] Michael R Dunlavey. Simulation of finite state machines in a quantum computer. *arXiv preprint quant-ph/9807026*, 1998.

[97] AC Cem Say and Abuzer Yakaryılmaz. Quantum finite automata: A modern introduction. In *Computing with New Resources*, pages 208–222. Springer, 2014.

[98] Edison Tsai and Marek Perkowski. A quantum algorithm for automata encoding. *Facta Universitatis, Series: Electronics and Energetics*, 33(2):169–215, 2020.

[99] Noah Linden and Sandu Popescu. The halting problem for quantum computers. *arXiv preprint quant-ph/9806054*, 1998.

[100] Stefano Guerrini, Simone Martini, and Andrea Masini. Quantum turing machines computations and measurements. *arXiv preprint arXiv:1703.07748*, 2017.

[101] Turlough Neary and Damien Woods. Small weakly universal turing machines. In *International Symposium on Fundamentals of Computation Theory*, pages 262–273. Springer, 2009.

[102] Hector Zenil, Liliana Badillo, Santiago Hernández-Orozco, and Francisco Hernández-Quiroz. Coding-theorem like behaviour and emergence of the universal distribution from resource-bounded algorithmic probability. *International Journal of Parallel, Emergent and Distributed Systems*, 34(2):161–180, 2019.

[103] Aritra Sarkar, Zaid Al-Ars, Carmen G Almudever, and Koen Bertels. An algorithm for dna read alignment on quantum accelerators. *arXiv preprint arXiv:1909.05563*, 2019.

[104] Vlatko Vedral, Adriano Barenco, and Artur Ekert. Quantum networks for elementary arithmetic operations. *Physical Review A*, 54(1):147, 1996.

[105] Nader Khammassi, Imran Ashraf, J v Someren, Razvan Nane, AM Krol, M Adriaan Rol, L Lao, Koen Bertels, and Carmen G Almudever. Openql: A portable quantum programming framework for quantum accelerators. *arXiv preprint arXiv:2005.13283*, 2020.

[106] Ryan LaRose. Overview and comparison of gate level quantum software platforms. *Quantum*, 3:130, 2019.

[107] Marcus Hutter. Algorithmic information theory: a brief non-technical guide to the field. *arXiv preprint cs/0703024*, 2007.

[108] Ray J Solomonoff. A formal theory of inductive inference. part i. *Information and control*, 7(1):1–22, 1964.

[109] Andrei Nikolaevich Kolmogorov. Three approaches to the quantitative definition of information. *International journal of computer mathematics*, 2(1-4):157–168, 1968.

[110] Gregory J Chaitin. On the length of programs for computing finite binary sequences. *Journal of the ACM (JACM)*, 13(4):547–569, 1966.

[111] Ming Li, Paul Vitányi, et al. *An introduction to Kolmogorov complexity and its applications*, volume 3. Springer, 2008.

[112] MTCAJ Thomas and A Thomas Joy. *Elements of information theory*. Wiley-Interscience, 2006.

[113] Leonid Anatolevich Levin. Laws of information conservation (nongrowth) and aspects of the foundation of probability theory. *Problemy Peredachi Informatsii*, 10(3):30–35, 1974.

[114] Hector Zenil, Santiago Hernández-Orozco, Narsis A Kiani, Fernando Soler-Toscano, Antonio Rueda-Toicen, and Jesper Tegnér. A decomposition method for global evaluation of shannon entropy and local estimations of algorithmic complexity. *Entropy*, 20(8):605, 2018.

[115] Leonid Anatolevich Levin. Universal sequential search problems. *Problemy peredachi informatsii*, 9(3):115–116, 1973.

[116] Charles H Bennett. *Logical depth and physical complexity*. Citeseer, 1988.

[117] Jürgen Schmidhuber. The speed prior: a new simplicity measure yielding near-optimal computable predictions. In *International conference on computational learning theory*, pages 216–228. Springer, 2002.

[118] Elliot Catt and Marcus Hutter. A gentle introduction to quantum computing algorithms with applications to universal prediction. *arXiv preprint arXiv:2005.03137*, 2020.

[119] Aritra Sarkar, Zaid Al-Ars, and Koen Bertels. Quantum accelerated estimation of algorithmic information. *arXiv preprint arXiv:2006.00987*, 2020.

[120] Jonathan Gorard. Some quantum mechanical properties of the wolfram model, 2020.

[121] Marcus Hutter. *Universal artificial intelligence: Sequential decisions based on algorithmic probability*. Springer Science & Business Media, 2004.

[122] Kenneth O Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35, 2019.

[123] Gregory J Chaitin. Meta math! the quest for omega. *arXiv preprint math/0404335*, 2004.

[124] Chiara Marletto. Constructor theory of life. *Journal of The Royal Society Interface*, 12(104):20141226, 2015.

[125] Yusuke Endoh. mame/doublehelix. `https://github.com/mame/doublehelix`, 2020.

[126] Aritra Sarkar, Zaid Al-Ars, and Koen Bertels. Quines are the fittest programs-nesting algorithmic probability converges to constructors. *Preprints*, 2020.

[127] Vincent Noireaux, Yusuke T Maeda, and Albert Libchaber. Development of an artificial cell, from self-organization to computation and self-reproduction. *Proceedings of the National Academy of Sciences*, 108(9):3473–3480, 2011.

[128] Moshe Sipper and James A Reggia. Go forth and replicate. *Scientific American*, 285(2):34–43, 2001.

[129] Anne Condon, Hélène Kirchner, Damien Larivière, Wallace Marshall, Vincent Noireaux, Tsvi Tlusty, and Eric Fourmentin. Will biologists become computer scientists? *EMBO reports*, 19(9):e46628, 2018.

[130] Hector Zenil and Peter Minary. Training-free measures based on algorithmic probability identify high nucleosome occupancy in dna sequences. *Nucleic acids research*, 47(20):e129–e129, 2019.

[131] Nicolas Gauvrit, Hector Zenil, Jean-Paul Delahaye, and Fernando Soler-Toscano. Algorithmic complexity for short binary strings applied to psychology: a primer. *Behavior research methods*, 46(3):732–744, 2014.

[132] Hector Zenil. Compression-based investigation of the dynamical properties of cellular automata and other systems. *arXiv preprint arXiv:0910.4042*, 2009.

[133] Hector Zenil and Elena Villarreal-Zapata. Asymptotic behavior and ratios of complexity in cellular automata. *International Journal of Bifurcation and Chaos*, 23(09):1350159, 2013.

[134] Olivier Brandouy, Jean-Paul Delahaye, Lin Ma, and Hector Zenil. Algorithmic complexity of financial motions. *Research in International Business and Finance*, 30:336–347, 2014.

[135] Hector Zenil and Jean-Paul Delahaye. An algorithmic information theoretic approach to the behaviour of financial markets. *Journal of Economic Surveys*, 25(3):431–463, 2011.

[136] Jean-Paul Delahaye and Hector Zenil. On the kolmogorov-chaitin complexity for short sequences. *ArXiv*, abs/0704.1043, 2007.

[137] Nicolas Gauvrit, Henrik Singmann, Fernando Soler Toscano, and Hector Zenil. Algorithmic complexity for short strings [r package acss version 0.2-5]. Available at https://cran.r-project.org/web/packages/acss/index.html, 2020.

[138] Cristian S Calude and Michael A Stay. Most programs stop quickly or never halt. *arXiv preprint cs/0610153*, 2006.

[139] Leslie G Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2):189–201, 1979.

[140] Scott Aaronson. Quantum computing, postselection, and probabilistic polynomial-time. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 461(2063):3473–3482, 2005.

[141] Scott Aaronson and Alex Arkhipov. The computational complexity of linear optics. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 333–342, 2011.

[142] Daniel S Abrams and Seth Lloyd. Simulation of many-body fermi systems on a universal quantum computer. *Physical Review Letters*, 79(13):2586, 1997.

[143] Han-Sen Zhong, Hui Wang, Yu-Hao Deng, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Jian Qin, Dian Wu, Xing Ding, Yi Hu, et al. Quantum computational advantage using photons. *Science*, 370(6523):1460–1463, 2020.

[144] Chu-Ryang Wie. Simpler quantum counting. *arXiv preprint arXiv:1907.08119*, 2019.

[145] Yohichi Suzuki, Shumpei Uno, Rudy Raymond, Tomoki Tanaka, Tamiya Onodera, and Naoki Yamamoto. Amplitude estimation without phase estimation. *Quantum Information Processing*, 19(2):75, 2020.

[146] Scott Aaronson and Patrick Rall. Quantum approximate counting, simplified. In *Symposium on Simplicity in Algorithms*, pages 24–32. SIAM, 2020.

[147] Zhang Jiang, Eleanor G Rieffel, and Zhihui Wang. Near-optimal quantum circuit for grover's unstructured search using a transverse field. *Physical Review A*, 95(6):062317, 2017.

[148] Mauro ES Morales, Timur Tlyachev, and Jacob Biamonte. Variational learning of grover's quantum search algorithm. *Physical Review A*, 98(6):062333, 2018.

[149] Stephen Wolfram. The physicalization of metamathematics and its implications for the foundations of mathematics. *arXiv preprint arXiv:2204.05123*, 2022.

[150] Alyssa Adams, Hector Zenil, Paul CW Davies, and Sara Imari Walker. Formal definitions of unbounded evolution and innovation reveal universal mechanisms for open-ended evolution in dynamical systems. *Scientific reports*, 7(1):1–15, 2017.

[151] Henri Atlan and Moshe Koppel. The cellular computer dna: program or data. *Bulletin of mathematical biology*, 52(3):335–348, 1990.

[152] Shan Dong and David B Searls. Gene structure prediction by linguistic methods. *Genomics*, 23(3):540–551, 1994.

[153] François Coste. Learning the language of biological sequences. In *Topics in grammatical inference*, pages 215–247. Springer, 2016.

[154] Edgar E Vallejo and Fernando Ramos. Evolving turing machines for biosequence recognition and analysis. In *European Conference on Genetic Programming*, pages 192–203. Springer, 2001.

[155] Rómulo Antão, Alexandre Mota, and JA Tenreiro Machado. Kolmogorov complexity as a data similarity metric: application in mitochondrial dna. *Nonlinear Dynamics*, 93(3):1059–1071, 2018.

[156] K Bertels, I Ashraf, R Nane, X Fu, L Riesebos, S Varsamopoulos, A Mouedenne, H Van Someren, A Sarkar, and N Khammassi. Quantum computer architecture: Towards full-stack quantum accelerators. *arXiv preprint arXiv:1903.09575*, 2019.

[157] JA Tenreiro Machado, João M Rocha-Neves, and José P Andrade. Computational analysis of the sars-cov-2 and other viruses based on the kolmogorov's complexity and shannon's information theories. *Nonlinear Dynamics*, 101(3):1731–1750, 2020.

[158] David E Gordon, Gwendolyn M Jang, Mehdi Bouhaddou, Jiewei Xu, Kirsten Obernier, Kris M White, Matthew J O'Meara, Veronica V Rezelj, Jeffrey Z Guo, Danielle L Swaney, et al. A sars-cov-2 protein interaction map reveals targets for drug repurposing. *Nature*, pages 1–13, 2020.

[159] Alyssa Adams. The role of emergence in open-ended systems. *AUTOMATA*, pages Slide 51, yet to be published, 2020.

[160] Christof Angermueller, David Belanger, Andreea Gane, Zelda Mariet, David Dohan, Kevin Murphy, Lucy Colwell, and D Sculley. Population-based black-box optimization for biological sequence design. *arXiv preprint arXiv:2006.03227*, 2020.

[161] Henry Gordon Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical society*, 74(2):358–366, 1953.

[162] Markus P Mueller. Law without law: from observer states to physics via algorithmic information theory. *Quantum*, 4:301, 2020.

[163] Pedro Domingos. *The master algorithm: How the quest for the ultimate learning machine will remake our world*. Basic Books, 2015.

[164] Ahmed Hallawa, Thorsten Born, Anke Schmeink, Guido Dartmann, Arne Peine, Lukas Martin, Giovanni Iacca, AE Eiben, and Gerd Ascheid. Evo-rl: evolutionary-driven reinforcement learning. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 153–154, 2021.

[165] Jürgen Schmidhuber. On learning how to learn learning strategies. *Fakultat Fur Informatik, Technische Universitat Munchen*, 1995.

[166] Ryan Levy, Di Luo, and Bryan K Clark. Classical shadows for quantum process tomography on near-term quantum computers. *arXiv preprint arXiv:2110.02965*, 2021.

[167] Jonathan Kunjummen, Minh C Tran, Daniel Carney, and Jacob M Taylor. Shadow process tomography of quantum channels. *arXiv preprint arXiv:2110.03629*, 2021.

[168] Aritra Sarkar, Zaid Al-Ars, and Koen Bertels. Estimating algorithmic information using quantum computing for genomics applications. *Applied Sciences*, 11(6):2696, 2021.

[169] Giacomo Torlai, Guglielmo Mazzola, Juan Carrasquilla, Matthias Troyer, Roger Melko, and Giuseppe Carleo. Neural-network quantum state tomography. *Nature Physics*, page 1, 2018.

[170] Matteo M Wauters, Emanuele Panizon, Glen B Mbeng, and Giuseppe E San-

toro. Reinforcement-learning-assisted quantum optimization. *Physical Review Research*, 2(3):033446, 2020.

[171] Jiahao Yao, Lin Lin, and Marin Bukov. Reinforcement learning for many-body ground state preparation based on counter-diabatic driving. *arXiv preprint arXiv:2010.03655*, 2020.

[172] Javier Rivera-Dean, Patrick Huembeli, Antonio Acín, and Joseph Bowles. Avoiding local minima in variational quantum algorithms with neural networks. *arXiv preprint arXiv:2104.02955*, 2021.

[173] Juan Carrasquilla and Giacomo Torlai. Neural networks in quantum many-body physics: a hands-on tutorial. *arXiv preprint arXiv:2101.11099*, 2021.

[174] Samuel Yen-Chi Chen, Chih-Min Huang, Chia-Wei Hsing, Hsi-Sheng Goan, and Ying-Jer Kao. Variational quantum reinforcement learning via evolutionary optimization. *arXiv preprint arXiv:2109.00540*, 2021.

[175] Mario Krenn, Mehul Malik, Robert Fickler, Radek Lapkiewicz, and Anton Zeilinger. Automated search for new quantum experiments. *Physical review letters*, 116(9):090405, 2016.

[176] Hans J Briegel and Gemma De las Cuevas. Projective simulation for artificial intelligence. *Scientific reports*, 2(1):1–16, 2012.

[177] Alexey A Melnikov, Hendrik Poulsen Nautrup, Mario Krenn, Vedran Dunjko, Markus Tiersch, Anton Zeilinger, and Hans J Briegel. Active learning machine learns to create new quantum experiments. *Proc. Natl. Acad. Sci. U.S.A.*, 115(6):1221–1226, 2018.

[178] Vedran Dunjko, Jacob M Taylor, and Hans J Briegel. Advances in quantum reinforcement learning. In *In 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 282–287, 2017.

[179] David Silver, Satinder Singh, Doina Precup, and Richard S Sutton. Reward is enough. *Artificial Intelligence*, page 103535, 2021.

[180] Laurent Orseau. Universal knowledge-seeking agents. *Theoretical Computer Science*, 519:127–139, 2014.

[181] Laurent Orseau, Tor Lattimore, and Marcus Hutter. Universal knowledge-seeking agents for stochastic environments. In *International conference on algorithmic learning theory*, pages 158–172. Springer, 2013.

[182] Marcus Hutter. Universal algorithmic intelligence: A mathematical top$\rightarrow$ down approach. In *Artificial general intelligence*, pages 227–290. Springer, 2007.

[183] John R Koza and John R Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[184] János Neumann, Arthur W Burks, et al. *Theory of self-reproducing automata*, volume 1102024. University of Illinois press Urbana, 1966.

[185] Jarn de Jong. Fault-tolerant quantum computation: Implementation of a fault-tolerant swap operation on the ibm 5-qubit device. Master's thesis, Delft University of Technology, 2019.

[186] Isaac L Chuang and Michael A Nielsen. Prescription for experimental determination of the dynamics of a quantum black box. *Journal of Modern Optics*, 44(11-12):2455–2467, 1997.

[187] Zhibo Hou, Jun-Feng Tang, Christopher Ferrie, Guo-Yong Xiang, Chuan-Feng Li, and Guang-Can Guo. Experimental realization of self-guided quantum process tomography. *Physical Review A*, 101(2):022317, 2020.

[188] Masoud Mohseni, Ali T Rezakhani, and Daniel A Lidar. Quantum-process tomography: Resource analysis of different strategies. *Physical Review A*, 77(3):032322, 2008.

[189] Scott Aaronson. Shadow tomography of quantum states. *SIAM Journal on Computing*, 49(5):STOC18–368, 2019.

[190] Hsin-Yuan Huang, Richard Kueng, and John Preskill. Predicting many properties of a quantum system from very few measurements. *Nature Physics*, 16(10):1050–1057, 2020.

[191] Konrad Zuse. *Calculating space.* Massachusetts Institute of Technology, Project MAC Cambridge, MA, 1970.

[192] Lucien Hardy. Quantum theory from five reasonable axioms. *arXiv preprint quant-ph/0101012*, 2001.

[193] Lluís Masanes and Markus P Müller. A derivation of quantum theory from physical requirements. *New Journal of Physics*, 13(6):063001, 2011.

[194] Philipp Andres Höhn. Quantum theory from rules on information acquisition. *Entropy*, 19(3):98, 2017.

[195] Melvin M Vopson. The mass-energy-information equivalence principle. *AIP Advances*, 9(9):095206, 2019.

[196] John Archibald Wheeler. *Information, physics, quantum: The search for links.* CRC Press, 2018.

[197] Seth Lloyd. *Programming the universe: a quantum computer scientist takes on the cosmos.* Vintage, 2006.

[198] David Deutsch. On wheeler's notion of "law without law" in physics. In *Between Quantum and Cosmos*, pages 583–591. Princeton University Press, 2017.

[199] Richard P Feynman. Simulating physics with computers. In *Feynman and computation*, pages 133–153. CRC Press, 2018.

[200] Susumu Katayama. Computable variants of aixi which are more powerful than aixitl. *arXiv preprint arXiv:1805.08592*, 2018.

[201] John Baez and Mike Stay. Algorithmic thermodynamics. *Mathematical Structures in Computer Science*, 22(5):771–787, 2012.

[202] Nicole Yunger Halpern, Naga BT Kothakonda, Jonas Haferkamp, Anthony Munson, Jens Eisert, and Philippe Faist. Resource theory of quantum uncomplexity. *arXiv preprint arXiv:2110.11371*, 2021.

[203] Artemy Kolchinsky and David H Wolpert. Thermodynamic costs of turing machines. *Physical Review Research*, 2(3):033312, 2020.

[204] Hector Zenil. Experimental algorithmic information theory | complexity and randomness. Available at http://www.mathrix.org/experimentalAIT/, 2020.

[205] Aritra Sarkar. Advanced-research-centre/qubio. https://github.com/Advanced-Research-Centre/QuBio/tree/master/Project_01/classical, 2020.

[206] Stephen Wolfram. Exploring rulial space: The case of turing machines. https://writings.stephenwolfram.com/2020/06/exploring-rulial-space-the-case-of-turing-machines/, 2020.

[207] Thomas Lubinski, Sonika Johri, Paul Varosy, Jeremiah Coleman, Luning Zhao, Jason Necaise, Charles H Baldwin, Karl Mayer, and Timothy Proctor. Application-oriented performance benchmarks for quantum computing. *arXiv preprint arXiv:2110.03137*, 2021.

[208] Frank Leymann and Johanna Barzen. The bitter truth about gate-based quantum algorithms in the nisq era. *Quantum Science and Technology*, 5(4):044007, 2020.

[209] Austin Gilliam, Stefan Woerner, and Constantin Gonciulea. Grover adaptive search for constrained polynomial binary optimization. *Quantum*, 5:428, 2021.

[210] Daochen Wang, Oscar Higgott, and Stephen Brierley. Accelerated variational quantum eigensolver. *Physical review letters*, 122(14):140504, 2019.

[211] Santiago Hernández-Orozco, Hector Zenil, Jürgen Riedel, Adam Uccello, Narsis A Kiani, and Jesper Tegnér. Algorithmic probability-guided machine learning on non-differentiable spaces. *Frontiers in artificial intelligence*, page 104, 2021.

[212] Giulio Chiribella and Daniel Ebler. Quantum speedup in the identification of cause–effect relations. *Nature communications*, 10(1):1–8, 2019.

[213] Cristhiano Duarte. Compatibility between agents as a tool for coarse-grained descriptions of quantum systems. *Journal of Physics A: Mathematical and Theoretical*, 53(39):395301, 2020.

[214] Adam R Brown and Leonard Susskind. Second law of quantum complexity. *Physical Review D*, 97(8):086015, 2018.

[215] Marcus Hutter. A complete theory of everything (will be subjective). *Algorithms*, 3(4):329–350, 2010.

[216] Rasmus Jaksland. Entanglement as the world-making relation: Distance from entanglement. *Synthese*, 198(10):9661–9693, 2021.

[217] Laurent Orseau and Mark Ring. Space-time embedded intelligence. In *International Conference on Artificial General Intelligence*, pages 209–218. Springer, 2012.

[218] Chris Fields, Donald D Hoffman, Chetan Prakash, and Robert Prentner. Eigenforms, interfaces and holographic encoding. *Constructivist Foundations*, 12(3):265–291, 2017.

[219] Alexander Jahn, Zoltán Zimborás, and Jens Eisert. Tensor network models of ads/qcft. *Quantum*, 6:643, 2022.

List of Publications

Applications of quantum computing

- Bertels, K., **Sarkar, A.**, & Ashraf, I. (2021). Quantum Computing—From NISQ to PISQ. *IEEE Micro, 41*(5), 24-32.
- Bertels, K., **Sarkar, A.**, Hubregtsen, T., Serrao, M., Mouedenne, A. A., Yadav, A., ... & Ashraf, I. (2020, March). Quantum computer architecture: Towards full-stack quantum accelerators. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (pp. 1-6). IEEE.
- Hubregtsen, T., Segler, C., Pichlmeier, J., **Sarkar, A.**, Gabor, T., & Bertels, K. (2020, March). Integration and Evaluation of Quantum Accelerators for Data-Driven User Functions. In *2020 21st International Symposium on Quality Electronic Design (ISQED)* (pp. 329-334). IEEE.

Quantum-accelerated genome sequence reconstruction

- **Sarkar, A.**, Al-Ars, Z., Almudever, C. G., & Bertels, K. L. (2021). QiBAM: Approximate Sub-String Index Search on Quantum Accelerators Applied to DNA Read Alignment. *Electronics, 10*(19), 2433.
- Krol, A. M., **Sarkar, A.**, Ashraf, I., Al-Ars, Z., & Bertels, K. (2022). Efficient decomposition of unitary matrices in quantum circuit compilers. *Applied Sciences, 12*(2), 759.
- **Sarkar, A.**, Al-Ars, Z., & Bertels, K. (2021). QuASeR: Quantum Accelerated de novo DNA sequence reconstruction. *Plos one, 16*(4), e0249850.

Quantum automata for algorithmic information

- **Sarkar, A.**, Al-Ars, Z., & Bertels, K. (2020). Quantum circuit design for universal distribution using a superposition of classical automata. *arXiv preprint arXiv:2006.00987.*
- **Sarkar, A.**, Al-Ars, Z., & Bertels, K. (2021). Estimating algorithmic information using quantum computing for genomics applications. *Applied Sciences, 11*(6), 2696.

Universal reinforcement learning in quantum environments

- **Sarkar, A.**, Al-Ars, Z., Gandhi, H., & Bertels, K. (2021). QKSA: Quantum Knowledge Seeking Agent – resource-optimized reinforcement learning using quantum process tomography. *arXiv preprint arXiv:2112.03643.*
- **Sarkar, A.**, Al-Ars, Z., & Bertels, K. (2020). Quines are the Fittest Programs - Nesting Algorithmic Probability Converges to Constructors. *Preprints 2020,* 2020100584.

CURRICULUM VITÆ

Aritra SARKAR

07-07-1991	Born in Purulia, India.

EDUCATION

2009–2013	Bachelor of Technology in Avionics Indian Institute of Space Science and Technology Thiruvananthapuram, India
2016–2018	Master of Science in Computer Engineering, cum laude Delft University of Technology Delft, The Netherlands
2018–2022	Ph.D., Quantum & Computer Engineering Delft University of Technology Delft, The Netherlands

Propositions

accompanying the dissertation

**APPLICATIONS OF
QUANTUM COMPUTATION AND ALGORITHMIC INFORMATION**
FOR CAUSAL MODELING IN GENOMICS AND REINFORCEMENT LEARNING

by

Aritra SARKAR

1. Philosophical aspects of interdisciplinary research lead to many valuable scientific insights.
2. Variational quantum heuristics share the 'correlation versus causation' problem with current machine learning models.
   ** This proposition pertains to Chapter 2 of this dissertation*
3. We need to understand the thermodynamic properties of mutating universal constructors to transcend to an intergalactic civilization.
   ** This proposition pertains to Chapter 3 of this dissertation*
4. The interference patterns we observe as shadows on Plato's classical cave help us to tell mathematical stories about Hilbert space.
   ** This proposition pertains to Chapter 4 of this dissertation*
5. Shadow libraries that provide open access to knowledge over intellectual property rights are best morally personified as Robin Hood rather than pirates.
6. All models require some axioms/assumptions/faith that define their limits.
7. Knowing everything is equivalent to knowing nothing.
8. Sisyphus (in the Absurd metaphor from Albert Camus) derives his happiness from discovering paths that are easier to remember and climb.
9. A proposition on self-referential proofs will always be opposed during the doctoral defense if a committee member thinks it is wrong.

These propositions are regarded as opposable and defendable, and have been approved as such by the promotor Prof. Dr. Koen L. M. Bertels and Dr. ir. Zaid Al-Ars.

9 789463 665308