How to Compute Using Quantum Walks

Viv Kendon
Physics Department
Durham University
Durham, UK
viv.kendon@durham.ac.uk

Quantum walks are widely and successfully used to model diverse physical processes. This leads to computation of the models, to explore their properties. Quantum walks have also been shown to be universal for quantum computing. This is a more subtle result than is often appreciated, since it applies to computations run on qubit-based quantum computers in the single walker case, and physical quantum walks in the multi-walker case (quantum cellular automata). Nonetheless, quantum walks are powerful tools for quantum computing when correctly applied. In this paper, I explain the relationship between quantum walks as models and quantum walks as computational tools, and give some examples of their application in both contexts.

1 Introduction

Quantum walks are widely appreciated for their mathematical intricacies and their many applications. Examples abound, from transport properties [36, 5] to quantum algorithms [46, 12]. There is an important difference between how quantum walks are used for computing and how they are used for modelling physical systems. For computing, obtaining an efficient algorithm is a key goal, whereas for models the goal is to accurately describe the physical characteristics of the system. Quantum walk experiments [40, 28, 6, 45] have mostly implemented physical quantum walks, with a walker (photon, atom) traversing a path in the experimental apparatus as the quantum walk evolves. There are a few implementations of quantum walks in an algorithmic setting, e.g., [44], where the walk is encoded into qubits that label the position of the walker. In this paper, I explain why this encoding step is critical for producing efficient quantum walk algorithms, and provide examples of algorithms that might be useful in the near future as quantum hardware develops. The paper is organised as follows. In section 2, I outline a general framework for reasoning about physical theories, science, engineering, and computing. This is followed in section 3 by discussion of what makes a computation "efficient". Section 4 contains a discussion of how quantum walks are used for efficient algorithms in a continuous-time setting. Section 5 summarises the discussion and considers the future of quantum walk computing.

2 A framework for physical computing

It is helpful to start by being more precise about what computing is. A *computation* is specified mathematically as a calcuation or process on input data. While there is not a clear and precise definition, fuzzy definitions are sufficient for this paper, where we are interested in the possibilities and constraints that the laws of physics place on computation. We can define a *computer* as a physical system used to carry out a specific computation. For example, a smart phone, your brain, an abacus – very different physical systems can all be computers. The origin of the term "computer" (up to about 1950) was humans doing calculations that were part of a larger overall calculation, so could be done faster in parallel. Something

G. Di Molfetta, V. Kendon & Y. Shikano (Eds.): 9th International Conference on Quantum Simulation and Quantum Walks (QSQW 2020)
EPTCS 315, 2020, pp. 1–17, doi:10.4204/EPTCS.315.1

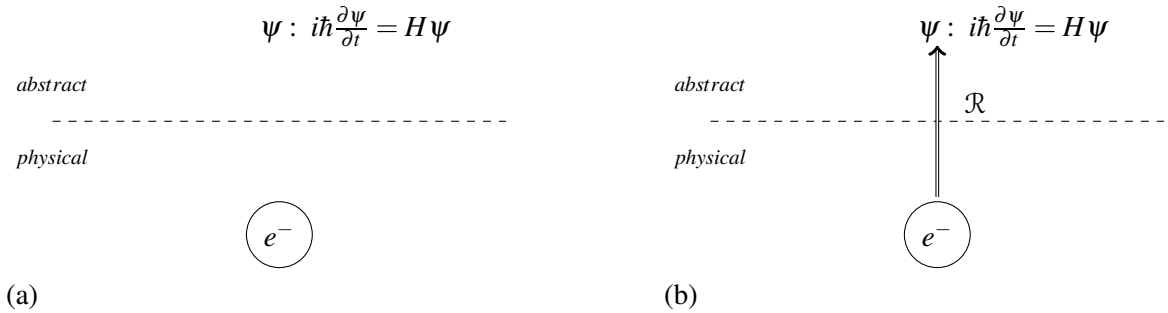

Figure 1: (a) Spaces of abstract and physical objects: here an electron e^- and a quantum mechanical wavefunction ψ. (b) The modelling representation relation $\mathcal{R}$ used to connect e^- to ψ.

we can immediately specify for all these different computers is that (successfully) carrying out an actual computation must produce an *output*. Any computation with no output can be replaced by a brick doing nothing. This effectively rules out pancomputationalism, in which everything in the universe is considered to be computing [41]. This is not a useful stance to take, there is then no distinction between being a computer and being any other type of physical system. A tighter definition and framework is proposed in [24], outlined here as follows.

2.1 Representation:

The basic building block of the framework is the relationship between a physical system and a mathematical description of it. An example is shown in figure 1, of an electron e^- and a wavefunction ψ describing the electron. These two different things, one physical, the other abstract, are related by what philosophers call the *representation relation* [42, 26], which we denote by $\mathcal{R}$. Actually, there are several different representation relations. We are going to use two of them, *modelling* and *instantiation*. In figure 1, we are using the representation relation $\mathcal{R}$ as the modelling relation, mediating from the *physical* to the *abstract*. Importantly, note that representations are not unique. The quantum mechanical wavefunction is one useful choice, but we might alternatively choose to represent an electron as a classical trajectory in classical mechanics, or as a point charge in electrostatics. The choice generally depends on what we happen to be interested in. Thus, representations are theory dependent, and henceforth we will indicate this by writing $\mathcal{R}_{\mathcal{T}}$ with the subscript $\mathcal{T}$ indicating the relevant theory.

There are many philosophical open questions about representation. We are going to leave the nuances of the open questions to the philosophers, and see what we can do with a framework based on abstract models of physical systems. Despite the apparent separation between the abstract and physical realms in the diagrams, this is not a dualist theory. Everything in the diagrams is physical, everything abstract has to be instantiated in a physical representation, be that neuronal activity in your brain, squiggles on paper, electrons in a computer, or patterns on a display screen. This implies that there has to be a "representational entity", that owns the representation of the abstract model. This entity does not need to be human, or conscious, or necessarily even alive in the biological sense [25, 48]. Note that we are assuming some form of philosophical realism – that there is a reality "out there" that we share and that persists when we don't observe it. This is a natural assumption for physicists, but if you don't agree with this, you arrive at different conclusions, which you are perfectly entitled to hold.

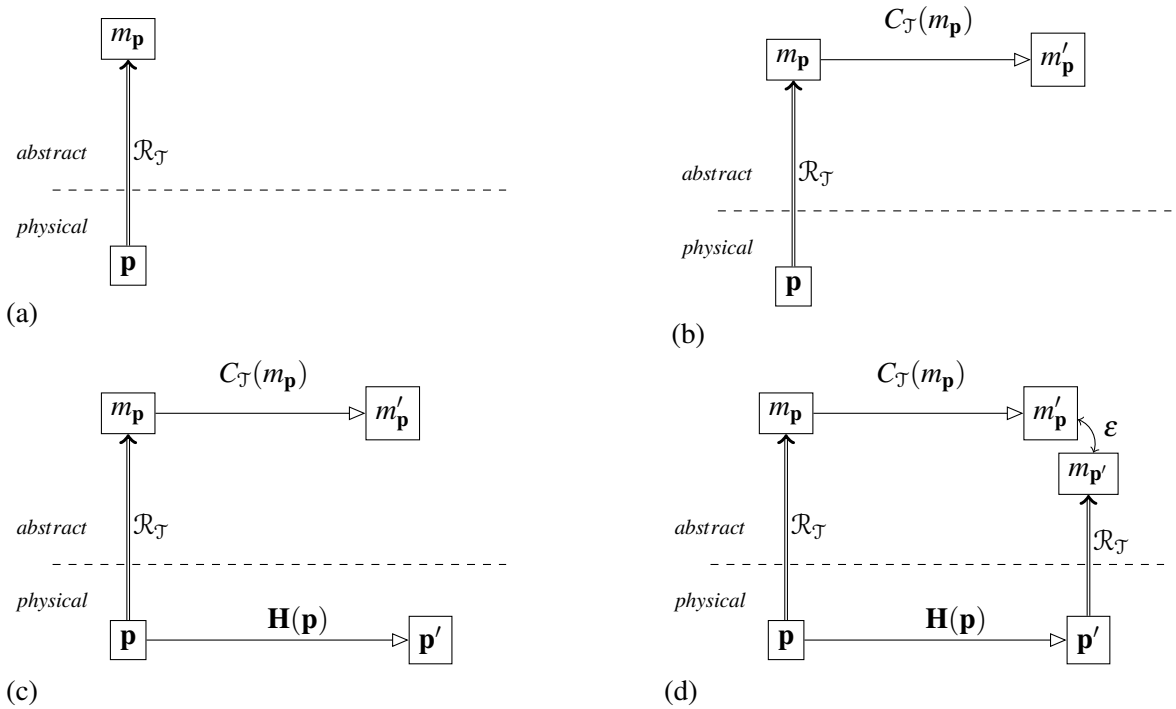

Figure 2: (a) Physical system $\mathbf{p}$ is represented abstractly by $m_{\mathbf{p}}$ using the modelling representation relation $\mathcal{R}_{\mathcal{T}}$ of theory $\mathcal{T}$. (b) Abstract dynamics $C_{\mathcal{T}}(m_{\mathbf{p}})$ give the evolved abstract state $m'_{\mathbf{p}}$. (c) Physical dynamics $\mathbf{H}(\mathbf{p})$ give the final physical state $\mathbf{p}'$. (d) $\mathcal{R}_{\mathcal{T}}$ is used again to represent $\mathbf{p}'$ as $m_{\mathbf{p}'}$, the difference between $m'_{\mathbf{p}}$ and $m_{\mathbf{p}'}$ is denoted ε.

2.2 The scientific process:

We can use the representation relation to describe how we do science, see figure 2. Given a physical object $\mathbf{p}$, we can use a theory $\mathcal{T}$ to represent the object as $m_{\mathbf{p}}$, using the representation relation $\mathcal{R}_{\mathcal{T}}$, as in figure 2 (a). Theory $\mathcal{T}$ says that the object will evolve under some dynamics $C_{\mathcal{T}}$ to become $m'_{\mathbf{p}}$, as in figure 2 (b). Experiment or observation of the physical object $\mathbf{p}$ as it evolves under its natural dynamics denoted $\mathbf{H}(\mathbf{p})$ to become $\mathbf{p}'$ is shown in figure 2 (c). We now wish to compare theory and experiment/observation. To do this, we use the representation relation again, on $\mathbf{p}'$, to obtain the abstract object $m_{\mathbf{p}'}$. We now have two objects of the same (mathematical) type, $m'_{\mathbf{p}}$ and $m_{\mathbf{p}'}$, that we can compare. There are, of course, many reasons why we don't expect perfect agreement between experiment and theory. The measurement precision in the experiment will be limited, the theory may not take into account all the possible errors, or we may know it is simpler than reality, but still a useful approximate theory. We denote the difference between theory and experiment by ε, as indicated in figure 2 (d). The goal of science is to develop theories that have a small enough ε to be useful to make predictions; for a "good" theory, $m'_{\mathbf{p}} \simeq m_{\mathbf{p}'}$. How we decide what makes a theory "good enough" is a major unsolved problem in the philosophy of science. We will just assume that what we do works, based on good evidence of the effectiveness of science, despite philosophers not agreeing on exactly how. Figure 2 is a simple building block in a much more complicated process. Science is a complex set of many interlocking diagrams like this, in which experimental equipment is repeatedly tested and characterised before being used for a new experiment.

Prediction is the process of using our current theories to extrapolate into areas where we don't yet

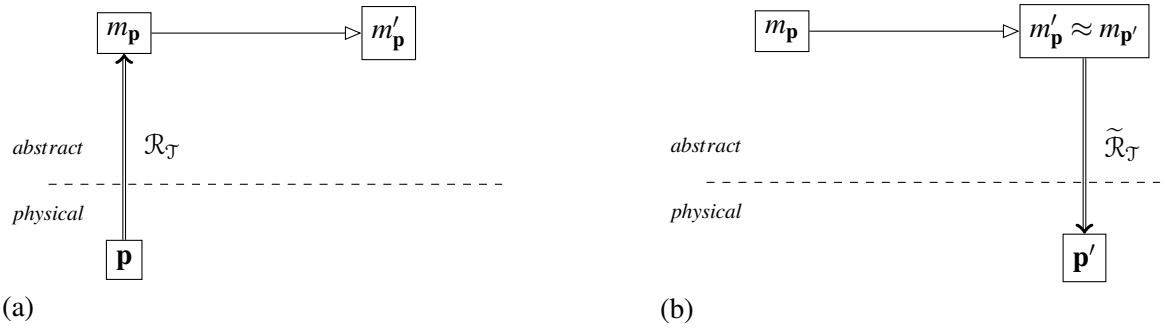

Figure 3: (a) The 'predict process': abstract theory is used to predict physical evolution. (b) The 'instantiation process': using an instantiation representation relation $\widetilde{\mathcal{R}}_{\mathcal{T}}$, a physical object is found corresponding to the predicted evolution.

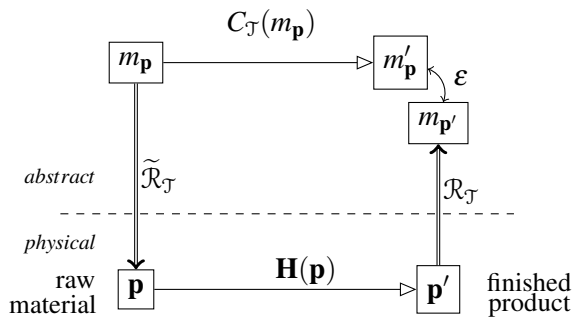

Figure 4: Making an artifact p' from a design specification $C_{\mathcal{T}}(m_{\mathbf{p}})$ and raw material $\mathbf{p}$.

have experimental observations. Thus, we can design new experiments, based on predictions from our existing theories. Figure 3 (a) shows how a good theory can be used to predict the outcomes of experiments that have not been done, including the possible existence of new particles or physical properties of systems.

Instantiation builds on prediction to locate or create an object predicted by theory, figure 3 (b). The instantiation representation relation $\widetilde{\mathcal{R}}_{\mathcal{T}}$ is in some sense the inverse of the modelling representation relation $\mathcal{R}_{\mathcal{T}}$, restricted to the class of objects that exist. It is possible to have theories of objects that do not actually exist, and it is through experiments that we discover whether or not a predicted object actually exists.

2.3 Engineering

Once we have done enough science to develop a robust theory, in which our predictions and experimental results agree to within a small ε, we can put our theory to work. Figure 4 shows how raw materials $\mathbf{p}$ are turned into finished products $\mathbf{p}'$ following a design specification $C_{\mathcal{T}}(m_{\mathbf{p}})$. The final step in the process is to check the quality of the production process by using the modelling representation relation $\mathcal{R}_{\mathcal{T}}$ on the finished product and comparing this with the design specification. Engineering, i.e., building technology, effectively reverses the modelling representation relation to instantiate the physical objects $\mathbf{p}'$ we want to make. It is important to remember that $\widetilde{\mathcal{R}}_{\mathcal{T}}$ is a shorthand for a lot of science, to enable us to manufacture things reliably.

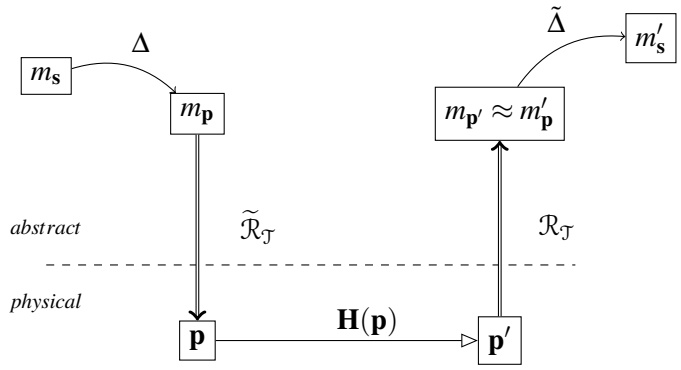

Figure 5: Carrying out a computation on a physical computer $\mathbf{p}$. The abstract problem $m_{\mathbf{s}}$ is first encoded Δ into the model of the computer $m_{\mathbf{p}}$. The computation is set up $\widetilde{\mathcal{R}}_{\mathcal{T}}$, and run, and the result read out $\mathcal{R}_{\mathcal{T}}$. It is then decoded $\tilde{\Delta}$ into the solution $m'_{\mathbf{s}}$.

The difference between science and technology in terms of the abstraction/representation theory framework is that, if the abstract and physical don't match well enough, i.e. , if ε is too large, then for science we need to improve the theory $\mathcal{T}$ until it describes our experiments well enough, but for technology we need to improve the production process for the physical system $\mathbf{p}'$ until it meets out design specification.

2.4 Computing

Among the many things we engineer are computers. Figure 5 illustrates the process of using a physical computer in terms of the framework. The abstract problem we are trying to solve (labeled $m_{\mathbf{s}}$ in figure 5) must first be encoded into the abstract machine model $m_{\mathbf{p}}$ using encoding Δ. Then, we can instantiate $m_{\mathbf{p}}$ in $\mathbf{p}$ using the reverse representation relation $\widetilde{\mathcal{R}}_{\mathcal{T}}$. The computation proceeds by evolving $\mathbf{p}$ into $\mathbf{p}'$ using $\mathbf{H}(\mathbf{p})$. When it has finished, we measure or observe the output state $\mathbf{p}'$, and represent it as model $m'_{\mathbf{p}}$. Now, we can assume $m'_{\mathbf{p}} \simeq m'_{\mathbf{p}}$, because the computer is based on sound science and has been engineered and tested to meet the design specification. Finally, we translate $m'_{\mathbf{p}}$ into m'_s using the appropriate decoding $\tilde{\Delta}$. The physical system $\mathbf{p}$ in figure 5 is representing an unrelated abstract computation via the mathematical encode Δ and decode $\tilde{\Delta}$ steps. In practice, there may be many layers of abstract encoding (known as "refinement" in computer science), e.g., compiling the source code of a computer program [24].

The key property of representation, from a computational point of view, is that the physical representation is arbitrary. It could have used a different instantiation just as well, different computers can carry out the same computation, and different programs can be written that carry out the same computation. Trivially, a different physical representation is obtained by inverting all the zeros and ones in the binary encoding in conventional computers. Less trivially, the number two can be represented in many ways, including $\{2$, II, (ii), (• •), two, ..$\}$. Group theory makes this idea mathematically precise, distinguishing between the concept of a set with certain properties, like "two-ness", and a particular representation of that group like (• •). A rather different example, of bacteria using the same signalling molecule but with different meanings in different species, is explained in detail in [25].

A further consequence of everything being physical is that a physical computation actually taking place includes a *computational entity*, a representational entity that owns the computation and where the abstract representation is physically instantiated. Hence, it is clear that a brain is a computer (among

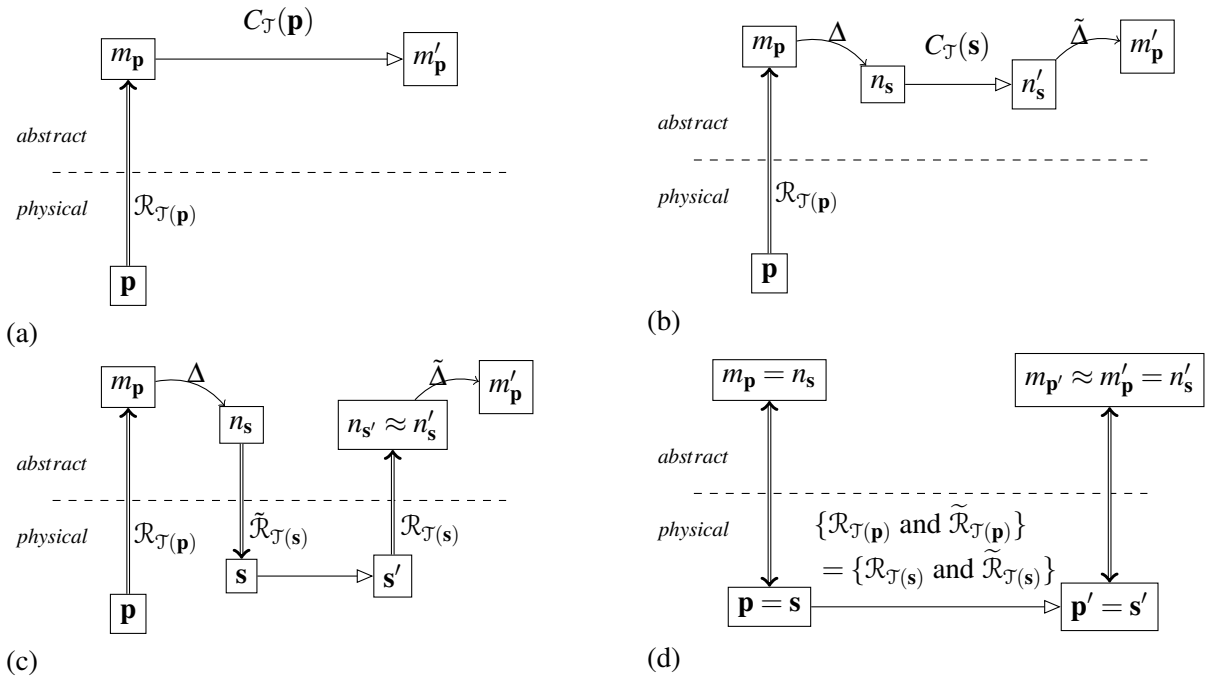

Figure 6: (a) Predict process using $C_{\mathcal{T}}(\mathbf{p})$ to find the abstract prediction for the evolution of system $\mathbf{p}$. (b) Encoding $C_{\mathcal{T}}(\mathbf{p})$ in the simulator model dynamics, $C_{\mathcal{T}}(\mathbf{s})$ using Δ and $\tilde{\Delta}$. (c) Adding the compute process: the physical simulator $\mathbf{s}$ now determines the abstract evolution $C_{\mathcal{T}}(\mathbf{s})$ and via encode/decode $\Delta/\tilde{\Delta}$ calculates $C_{\mathcal{T}}(\mathbf{p})$. (d) A system simulating itself, with encode and decode as identities. Note that the models $m_{\mathbf{p}}$ and $n_{\mathbf{s}}$ must be present for this to be simulation.

other functions). However, humans not are not required: a robot could use a calculator; a bacterium has no brain [25]. Those familiar with communications may have met Alice and Bob in presentations of communications theory or experiments. Alice and Bob serve to remind us that communications involves the transmission of information between entities for which the information is meaningful in some way. While we can do the theory entirely abstractly, what makes it communication rather than mathematics is the possibility of transmitting meaningful information between representational entities.

2.5 Simulation

Simulation is calculating the details of a model of a physical system. We can only simulate our models, it does not make sense to say we are directly simulating a physical system. Working directly with a physical system is an experiment, or engineering. Simulation is thus a particular type of computation that happens to be calculating something about the theory for a physical system. The diagrams for system $\mathbf{s}$ running as a simulator for system $\mathbf{p}$, constructed as a compute process nested within a predict process, are given in figure 6. Figure 6 (a) is the normal prediction process, where a model $m_{\mathbf{p}}$ is used to predict the properties of $\mathbf{p}$. Figure 6 (b) is an intermediate step, where the model of the simulator (computer) $m_{\mathbf{s}}$ is used to set up the encode Δ and decode $\tilde{\Delta}$ steps, and the program $C_{\mathcal{T}}(\mathbf{s})$ to run the simulation. Figure 6 (c) shows the simulation taking place, with the program $C_{\mathcal{T}}(\mathbf{s})$ running on the physical simulator $\mathbf{s}$. Note that the theories in $C_{\mathcal{T}}(\mathbf{s})$ and $C_{\mathcal{T}}(\mathbf{p})$ do not need to be the same. They have been shown as the same in figure 6 to avoid cluttering the notation. A computer can simulate itself, see figure 6 (d), and

Table 1: Relationship between unary and binary encoding of numbers.

Number	*Unary*	*Binary*
0		0
1	•	1
2	••	10
3	•••	11
4	••••	100
...	...	...
8	•••••••••	1000
...	...	...
N	$N \times \bullet$	$\log_2 N$ bits
...	...	...
example	database records	index of records

this is often done, as *virtual machines*. In figure 6 (d), the self-simulation is shown with the encode and decode as the identity. For virtual machines, the encode and decode are usually not the identity, with the physical computer simulating one (or more than one) approximate copy of itself. However, most physical systems do not simulate themselves in real time. Unless there is representation, there is no theory for what is being calculated, and no computational or representational entity for which the computation is meaningful. Computation as defined in this framework is a high level process that must involve representation [24, 25].

3 Efficient computation

In the abstraction/representation theory framework just outlined, computing is defined as a high level process that involves engineered (or evolved) objects and computational entities. The requirement for a computational entity implies computing needs to be purposeful, or useful. Which implies that, in general, computing is done in order to efficiently solve a problem. We now turn to the question of what is needed for efficient computing, what it is that makes one computer better than another for a particular computation. There are many facets to this question and we will only consider a few key points that are important for quantum computing.

The first important consideration is that *efficient* has different criteria for different purposes. On the one hand, complexity theorists consider asymptotic scaling of computations. For them, efficient is usually defined as a polynomial scaling of the computational resources with respect to the problem size. On the other hand, for practical computation, *efficient* means obtaining answers on the timescales relevant for the computational entity. For rendering graphics, the timescale is faster than the eye detects flickers. For hard simulations, days or weeks are acceptable. For real-time monitoring or prediction, it means faster than the physical process being modelled: an accurate weather forecast is not nearly so useful the day after the weather has already happened. For quantum computing, the complexity of a problem and the predicted quantum advantage are a good starting point, but not the whole story when it comes to building a useful quantum computer. A more favourable prefactor in the scaling can be enough to make using a quantum computer worthwhile, especially for time-sensitive computation. It isn't necessary to solve a larger problem than is possible classically to gain an advantage. Solving a problem faster can be enough to gain a competitive advantage, depending on the value of the solution.

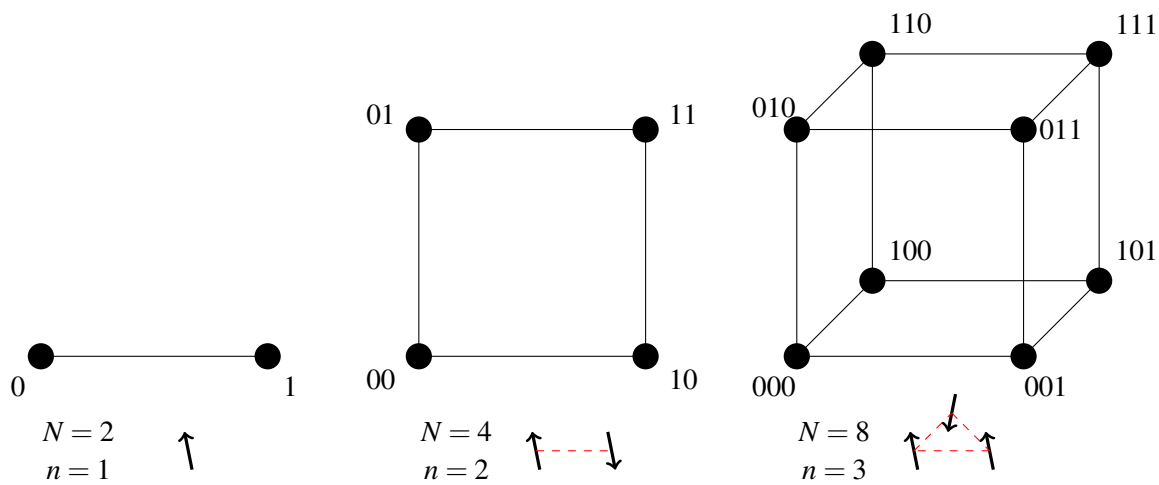

Figure 7: Hypercubes and qubits for $n = 1, 2, 3$, showing the encoding from quantum walk graph (upper) of size $N = 2^n$ to qubit graph (lower) of size n. The red dotted lines indicate possible interaction terms between qubits, that can be used to encode problems into Hamiltonians.

A second important consideration is efficient encoding of the data. Binary encoding is one reason why classical digital computers work so efficiently. Table 3 shows how binary encoding is exponentially more efficient in terms of memory use compared with unary. This becomes increasingly important as the size of the numbers increases. Modern digital computers use even more efficient representations, such as floating point numbers, which provide a trade off between efficiency and precision.

The same encoding considerations apply when building quantum computers [18, 4]. In addition, because the measurement process in quantum mechanics is non-trivial and can alter the state of the register, the encoding has an impact on the efficiency and accuracy of the measurement process. For a unary encoding, it is necessary to measure in a way that can distinguish between N different outcomes. This becomes increasingly challenging as N becomes large. For binary encoding, $\log_2 N$ measurements with two outcomes each is enough to distinguish N numbers. This is more efficient and better for accuracy, since each measurement only has to distinguish between two orthogonal outcomes. Doubling the size of the problem (N) only requires one more binary measurement, which is more efficient than doubling the number of measurement outcomes that must be distinguished.

In a quantum walk setting, a useful example of the efficiency of encoding is a quantum walk on a hypercube, see figure 7. A quantum walk on the vertices of an n-dimensional hypercube can be encoded into n qubits. An n-dimensional hypercube has $N = 2^n$ vertices, which are labelled by the bitstrings of the qubit basis states, as shown in figure 7. The relationship between the graph in Hilbert space representing the graph on which the quantum walk takes place, and the graph formed by the qubits storing the labels of the vertices, which exist in real physical space, is the same as the relationship between unary and binary encoding in table 3. Quantum walks on hypercube graphs are especially useful for quantum computing, as will be explained in the next section. The efficiency of encoding into qubits is not just for hypercube graphs. Since any graph can have its vertices labeled with binary numbers, efficiently computing the evolution of the quantum walk on the graph can be done using the labels encoded into qubits. A physical quantum walk is not an efficient way to compute how the quantum walk evolves. This is easily illustrated by noting the ease of computing thousands or millions of steps of a quantum walk on a line using a classical digital computer. This is far beyond any physical quantum walk experiments

currently realised. However, fifty or more logical qubits would be enough to calculate a quantum walk beyond the capabilities of classical computers. Fifty qubits is not far beyond the capacity of current quantum computers from IBM or Google, although they are currently too noisy to carry out a long enough quantum computation.

4 Quantum computing with quantum walks and related models

Important results from Childs et al. [11, 35], proved that quantum walks are universal for quantum computing. However, this result for single walkers applies strictly only to quantum walks encoded into qubits. Without this encoding, the quantum walks are not efficient in the sense of complexity theory scaling with problem size, and thus cannot provide a quantum advantage. A second result from Childs et al. [15] proving that multiple interacting quantum walkers provide universal quantum computing does not require encoding into qubits. In this case, the physical quantum walks are efficient, equivalent to quantum cellular automata [49]. Boson sampling [1] is halfway betweem the two, consisting of multiple non-interacting quantum walkers. Boson sampling does not require encoding into qubits, but it is not universal for quantum computing. It can still solve certain hard problems, such as approximating the permanent of a complex matrix, more efficiently than the best known classical algorithm.

There are several quantum walk algorithms with complexity proofs of their speed up over classical algorithms for the same problem. Searching in an unsorted database is the first such algorithm [46]. Quantum walk search is now a workhorse subroutine for theorists constructing algorithms for more complex problems, providing a quadratic speed up over classical. For an unstructured database, random guessing in the best classical strategy available, which scales linearly with the problem size N, while quantum search scales as $\sqrt{N}$. The "glued trees" algorithm [12], provides an exponential speed up with respect to an oracle, but does not lead to applications in the same way as quantum search. Both search and the glued trees algorithm are permutation symmetric problems. Permutation symmetric problems are useful for testbed algorithms, to test the performance of quantum hardware. Problems which can be solved analytically, efficiently computed classically (assuming some knowledge of the solution), and implemented efficiently on a quantum computer, support detailed testing of the performance of quantum hardware. The implementation on quantum hardware can be efficiently achieved using extra qubits configured as gadgets to enforce the permutation symmetry [17].

4.1 Quantum computing in continuous time

We can conveniently represent quantum computing in a flattened version of the AR diagram,

$$\text{input} \longrightarrow \boxed{\text{encode}} \longrightarrow |\psi_{in}\rangle \longrightarrow \hat{U} \longrightarrow |\psi_{out}\rangle \longrightarrow \boxed{\text{decode}} \longrightarrow \text{result},$$

which shows the steps to encode a problem into a quantum computer, and carry out the computation by evolving the quantum state $|\psi_{in}\rangle$ using unitary operation $\hat{U}$ to produce $|\psi_{out}\rangle$. More generally, $\hat{U}$ can include interaction with the environment; this can also be modelled as unitary interactions in a larger Hilbert space that includes the environment. The unitary transformation $\hat{U}$ can be done by a sequence of quantum gates in the circuit model. A different approach is to engineer a Hamiltonian $\hat{H}(t)$ such that

$$|\psi_{out}\rangle = \exp\{-i\hbar \int dt\, \hat{H}(t)\}\, |\psi_{in}\rangle \tag{1}$$

The motivation for considering this form of processing for quantum computing is as follows. Classical bits are either zero or one, and can flip between the two values. On the other hand, qubits can be any

superposition: $\alpha|0\rangle + \beta|1\rangle$, and can change smoothly from zero to one or anything in between. Hence, discrete gates make sense for bits: bit-flip is all you can do. But for qubits, exact bit-flip is as hard as other rotations. Processing in continuous-time evolution thus makes sense for qubits. This is related to work on the foundations of quantum mechanics by Lucien Hardy [22]. The axiom that distinguishes quantum from classical in a discrete state space is the possibility of evolving smoothly from one state to another, instead of undergoing a discontinuous change.

There are several ways to quantum compute in continuous time: quantum walks; adiabatic quantum computing; quantum annealing; and special purpose quantum simulation. Each is described briefly in turn in the following subsections, apart from quantum simulation, for which see [7] for a detailed review.

4.2 Computing by quantum walk

First introduced in the continuous-time form by Farhi and Gutmann [20], and used for an algorithm with exponential speed up by Childs et al. [12]. Continuous-time quantum walks take place in a Hilbert space corresponding to a graph G, with basis states $|j\rangle$ for the label j of each vertex of the graph. If the graph has adjacency matrix $\mathbf{A}$, defined as $A_{jk} = 1$ iff $\exists$ an edge between vertices j and k, the Laplacian of the graph is defined by $\mathbf{L} = \mathbf{A} - \mathbf{D}$, where $\mathbf{D}$ is diagonal matrix $\mathbf{D}_{jj} = deg(j)$, i.e. the degree of vertex j of the graph. For undirected graphs, $\mathbf{L}$ is a symmetric matrix, and can be used to define the quantum walk Hamiltonian $\hat{H}_G$ such that $\langle j|\hat{H}_G|k\rangle = -\gamma\mathbf{L}_{jk}$, where γ is the transition rate (probability of moving to a connected vertex per unit time). The quantum walk evolves the initial state $|\psi(0)\rangle$ according to the Schrödinger equation, giving

$$|\psi(t)\rangle = \exp\{-i\hat{H}_G t\}\,|\psi(0)\rangle\,. \tag{2}$$

By itself, this doesn't provide an algorithm. There are two ways to obtain a quantum algorithm from this type of quantum walk. One way is to use a specific graph that defines the problem. This is the approach taken in the "glued trees" algorithm [12], where the problem is to find the exit node from the entrance node (the roots of the trees). The second way is to combine the quantum walk Hamiltonian with another Hamiltonian that specifies the problem. This is the approach taken in the search algorithm [46], where the marked state is treated differently from the rest of the vertices in the graph. In all cases, the quantum walk must be encoded into qubits, as explained in section 3, in order to obtain an efficient quantum algorithm. As an example of this encoding, the quantum walk on a hypercube graph of size n has $N = 2^n$ vertices, but can be computed using n qubits. The Hamiltonian is given by

$$\hat{H}_h = \gamma\left(n\mathbb{1} - \sum_{j=1}^{n}\sigma_j^{(x)}\right), \tag{3}$$

where $\sigma_j^{(x)}$ is the Pauli-x operator acting on the jth qubit. The $\sigma_j^{(x)}$ operators flip the jth qubit, equivalent to walking along the edges in the jth direction on the hypercube, see figure 7.

4.3 Adiabatic quantum computing

Introduced also by Farhi and Gutmann [19] as natural way to solve optimisation problems such as boolean satisfiability (SAT) problems. The idea is to encode the solution into the ground state of a quantum Hamiltonian $\hat{H}_{\text{problem}}$. Methods are known to do this efficiently [16, 34]. Then, starting in an easy-to-prepare Hamiltonian ground state of $\hat{H}_{\text{simple}}$, evolve the Hamiltonian from $\hat{H}_{\text{simple}}$ to $\hat{H}_{\text{problem}}$ smoothly,

$$\hat{H}_{AQC}(t) = \{1 - s(t)\}\hat{H}_{\text{simple}} + s(t)\hat{H}_{\text{problem}} \tag{4}$$

with $s(0) = 0$, $s(t_f) = 1$. If the time evolution is slow enough, the adiabatic theorem states that the quantum system will stay in the instantaneous eigenstate – in this case the ground state – throughout the time evolution with high probability. The key question is thus, how slow is slow enough. The speed is in general determined by the minimum gap between ground state and excited states, which you don't know, because calculating it is equivalent to solving the problem. There are many subtleties to how slow is "slow enough", but since we design the problem Hamiltonian, we can avoid the more pathological cases where the adiabatic theorem fails.

Adiabatic quantum computing as been controversial, since it looks like can solve NP-hard problems, and it would be surprising if it could do this efficiently from what we know about complexity theory. However, it is now clear it is likely to require exponential time, so would not be "efficient" according to theoretical CS. Nonetheless, any speed up is potentially useful in practice, so adiabatic quantum computing is still very much of interest in a practical setting. Not least, this is because it has been proved to be equivalent to digital quantum computation, that is, it can solve all the same problems using equivalent resources [2, 29]. It is also potentially more resistent to some errors [13], although the scaling and propagation of errors is not known in general in this setting.

Testbed implementation of adiabatic quantum computing has been done in NMR [47], but larger scale devices tend to be too noisy currently to be effective for adiabatic quantum computing. Introduced by Finilla et al, [21], quantum annealing is a noisy version of adiabatic quantum computing, with a low temperature bath that helps to remove energy to reach the ground state solution. Quantum tunneling helps to avoid the annealing process becoming stuck in false minima. The transverse Ising model is a particularly natural setting for quantum annealing [27]. Note that in the literature, the terms *adiabatic quantum computing* and *quantum annealing* are used interchangeably, which can lead to confusion: they are in fact two different computational models. While adiabatic quantum computing is what D-Wave Systems Inc. would like to be doing, what they are actually doing is closer to quantum annealing, with a lot of other types of unwanted noise, as well as a cold environment assisting the computation.

4.4 Hybrid QW-AQC algorithms

The reason for mentioning adiabatic quantum computing in a paper about quantum walks will become clear with an example. Quantum search algorithms are well-studied and well-understood in both discrete (gate model, quantum walks) and continuous-time settings. The search problem is to find the marked state in an unsorted database. In a continuous-time setting, the problem Hamiltonian is simple to define,

$$\hat{H}_p = \hat{H}_m = \mathbb{1} - |m\rangle\langle m|. \tag{5}$$

This works by making the marked state $|m\rangle$ one unit of energy lower than the rest of the states. In terms of Pauli operators on qubits, this does not look so simple,

$$\hat{H}_m = \mathbb{1} - \frac{1}{2^n}\prod_{j=1}^{n}(\mathbb{1} + q_j\sigma_j^{(z)}), \tag{6}$$

where $q_j \in \{-1, 1\}$ defines the bitstring corresponding to m for $-1 \equiv 0$ to convert to bits, and $\sigma_j^{(z)}$ is the Pauli-z operator acting on the jth qubit. There are techniques using extra qubits and perturbation theory that can implement this and other permutation symmetric problem Hamiltonians [17].

To implement a continuous-time quantum search algorithm, the full Hamiltonian is

$$\hat{H}_{QWS} = \hat{H}_G + \hat{H}_m, \tag{7}$$

where $\hat{H}_G$ is the quantum walk Hamiltonian on graph G. If the hypercube graph is used, $\hat{H}_G = \hat{H}_h$, as given in equation (3).

The initial state of the qubits is

$$|\psi(t=0)\rangle = 2^{-n/2} \sum_{j=0}^{2^n-1} |j\rangle = 2^{-n/2}(|0\rangle + |1\rangle)^{\otimes n} \tag{8}$$

which is the superposition of all possible states, and also happens to be the ground state of $\hat{H}_h$. This is a natural choice for an unbiased initial state, it faithfully represents our lack of knowledge of the identity of the marked state. Applying the quantum walk search Hamiltonian $\hat{H}_{QWS}$ to the initial state for a time $t_f \simeq \pi\sqrt{N}/2$ results in a final state $\left|\psi(t_f)\right\rangle \simeq |m\rangle$. A measurement of the qubits produces the index m of the marked state with high probability, thus solving the search problem in time $O(\sqrt{N})$, a quadratic improvement over the classical best strategy.

The adiabatic quantum search algorithm looks quite similar to the quantum walk search algorithm,

$$\hat{H}_{AQC}(t) = A(t)\hat{H}_h + B(t)\hat{H}_m \tag{9}$$

where $A(t)$ and $B(t)$ specify the time dependent proportions of the two component Hamiltonians, and satisfy $A(t_f) = B(0) = 0$, $A(0) = B(t_f) = 1$ and $0 \leq A(t), B(t) \leq 1$ for $0 \leq t \leq t_f$. As for the quantum walk, the qubits start in the initial state in equation (8), the ground state of $\hat{H}_h$, and are measured after a time $t_f \propto \sqrt{N}$ to obtain a quadratic quantum speed up.

Both quantum walk and adiabatic quantum search algorithms are analytically solvable [14, 43] in the large N limit, because they reduce to a two state single avoided crossing model [14, 39]. There are many subtleties involved in setting the parameters $A(t)$, $B(t)$, γ, to achieve a quantum speed up, for detailed discussion and results, see [39].

The important observation for this paper is that quantum walk search and adiabatic quantum search are special cases of the same algorithm, with different choices of $A(t)$ and $B(t)$ in equation (9) giving either quantum walk or adiabatic. The computational mechanisms are different in each case. For quantum walk, the system oscillates between the equal superposition initial state, and the marked state, while for adiabatic, the system transitions steadily from the initial state to the marked state. A smooth interpolation between adiabatic and quantum walk can be done, and optimal contributions from both mechanisms employed to improve performance on imperfect hardware [39].

Identifying the connections between quantum walk computing and adiabatic quantum computing allows several useful generalisations to be explored that are suitable for implementation on early quantum computers. Any algorithm that has been shown to work in an adiabatic setting is likely to be promising in a quantum walk setting, too. An example is finding the ground state of Sherrington-Kirkpatric spin glasses, for which adiabatic quantum computing is known to compare well against (classical) simulated annealing [27]. Callison et al [8] recently showed that quantum walks can find the ground state better than quantum search algorithms, using many repeated short runs. Quantum walks are potentially easier to implement experimentally, given their time-independent Hamiltonians. They are effective when used with many short runs and repeats, which suits early, noisy quantum hardware. The benefits of combining both quantum walk and adiabatic techniques can be demonstrated in a range of settings, especially in the presence of hardware imperfections [39]. However, it is important to use quantum algorithms where they will be most effective, to obtain a practical quantum advantage.

4.5 Hybrid classical-quantum algorithms

A quantum speed up using a quantum walk algorithm does not necessarily mean a quantum advantage overall for the computational task. For search in an unsorted database, quadratic is the best possible speed up, and has been proven to give an advantage over the bext possible classical algorithm [3]. However, the best classical algorithm is likely to be significantly better than random guessing, if the problem has some structure or correlations that can be exploited. Real world problems always have such correlations, this is what makes them interesting and non-trivial to solve. In the case of the spin glass ground state problem mentioned in section 4.4, the best known classical algorithm [23] is actually better than the pure quantum walk algorithm of [8]. However, a hybrid discrete-time quantum walk-classical algorithm that beats both has been presented by Montanaro [38]. This is an example of one route to obtaining a quantum advantage. Starting with the best classical algorithm, locate the bottleneck subroutine, and if amenable to a quantum speed up, replace it by a quantum version of the subroutine. Another example also from Montanaro [37] applies this technique in the setting of backtracking algorithms. Quantum walk search subroutines provide a further quadratic speed up compared with the classical algorithm. More examples of how to construct hybrid algorithms, in a continuous-time setting, can be found in [10, 9].

Employing quantum computers for the bottenecks in classical algorithms fits naturally into the diversification that is taking place currently in computer hardware. Since we have reached the limits of standard silicon-based computer chips in terms of their speed of operation, computer chips have become more specialised, to speed up computation of commonly used tasks. Graphics co-processors are one of the most widespread examples. They started out efficiently updating the screen image in real time, and graduated to computational science as massively parallel chips using shared memory. Today's computers also typically include a dedicated chip for communications tasks such as ethernet and wifi. A typical high performance computing facility now includes GPUs and field programmable gate arrays (FPGAs) as well as many conventional CPU cores. A quantum co-processor is a natural extension of this trend, and will enable hybrid quantum-classical algorithms to be run on well-integrated hardware.

5 Outlook

Quantum walks are widely used as models for physical processes, especially transport properties. They are also a useful tool for quantum algorithms. It is important to understand the different requirements for different uses of quantum walks. Mathematical models of physical processes are abstract constructions, their usefulness depends on comparision with experiments on physical systems, as elaborated in section 2.2. If the model does not describe the experimental results well enough, the model needs to be improved. These mathematical models can then be explored using computation, especially if the analytical solutions are complicated or intractable. A simple quantum walk on the line is easy to compute classically for large numbers of steps (thousands or millions). As quantum walks spread ballistically (linearly in time), the computational resources required grow linearly, and this scaling is generally considered to be efficient from a computational point of view. For quantum walks used as a computational tool, different criteria apply for when we might gain a computational advantage from quantum over classical. Efficient computation demands that the computation is encoded efficiently, and for quantum walks with single walkers, this means binary encoding into qubits (or equivalent) is essential. A physical quantum walk with a single walker cannot provide efficient quantum computing. Nonetheless, used correctly, quantum walks are a powerful computational tool.

Continuous-time quantum walks are part of a family of related continuous-time computational models, including: adiabatic quantum computing; quantum walk; quantum annealing; special purpose quan-

tum simulation. These computational models all use qubits for efficient binary encoding with continuous-time evolution to exploit the natural time-evolution of quantum mechanics. The addition of cooling through a low temperature environment can offset some of the damaging effects of noise on the quantum computation. Relating all these different computational models allows us to try different models and combinations to find the best for each particular problem. Especially interesting is to try quantum walks on problems so far only examined in an adiabatic quantum computing setting.

There are a few caveats to be aware of. Increasing numbers of qubits require more precise Hamiltonian parameters to be specified in the controls for the quantum computer, e.g., the coupling strengths between pairs of qubits in the problem Hamiltonian. There is evidence from D-Wave that they are reaching that limit at around 2000 qubits. However, 2000 qubits is still far more than classical computers can simulate, and the limit could be higher for other architectures. There is also as yet no fully developed theory of error correction in this continuous-time setting, although quantum error correcting codes can be used to to provide robustness [33], and NMR and other quantum control techniques can improve the fidelity of the time evolution. Even modest numbers of qubits can be used as quantum co-processors for bottleneck subroutines. This is a natural way to exploit quantum computers in the early stages of development, through hybrid algorithms that use quantum computing only for the steps that are hardest for classical computers to do efficiently [10, 9]. More work is needed to develop theories of hybrid computational hardware, to enable the combinations to be fully exploited [30, 31].

Acknowledgements

This work was supported by the UKRI Engineering and Physical Science Council fellowship grant number EP/L022303/1. Thanks to Jemma Bennett and Stephanie Foulds for interesting discussions and checking the manuscript. A recording of the presentation of this work is available at [32].

References

[1] Scott Aaronson & Alex Arkhipov (2013): *The Computational Complexity of Linear Optics*. Theory of Computing 9(4), pp. 143–252, doi:10.4086/toc.2013.v009a004.

[2] Dorit Aharonov, Wim van Dam, Julia Kempe, Zeph Landau, Seth Lloyd & Oded Regev (2007): *Adiabatic Quantum Computation is Equivalent to Standard Quantum Computation*. SIAM Journal on Computing 37, pp. 166–194, doi:10.1137/S0097539705447323.

[3] C. Bennett, E. Bernstein, G. Brassard & U. Vazirani (1997): *Strengths and Weaknesses of Quantum Computing*. SIAM Journal on Computing 26(5), pp. 1510–1523, doi:10.1137/S0097539796300933.

[4] Robin Blume-Kohout, Carlton M. Caves & Ivan H. Deutsch (2002): *Climbing Mount Scalable: Physical Resource Requirements for a Scalable Quantum Computer*. Found. Phys. 32(11), pp. 1641–1670, doi:10.1023/A:1021471621587.

[5] Hamza Bougroura, Habib Aissaoui, Nicholas Chancellor & Viv Kendon (2016): *Quantum-walk transport properties on graphene structures*. Phys. Rev. A 94, p. 062331, doi:10.1103/PhysRevA.94.062331.

[6] M. A. Broome, A. Fedrizzi, B. P. Lanyon, I. Kassal, A. Aspuru-Guzik & A. G. White (2010): *Discrete single-photon quantum walks with tunable decoherence*. Phys. Rev. Lett. 104, p. 153602, doi:10.1103/PhysRevLett.104.153602.

[7] K. L. Brown, W. J. Munro & V. M. Kendon (2010): *Using Quantum Computers for Quantum Simulation*. Entropy 12(11), pp. 2268–2307, doi:10.3390/e12112268.

[8] A Callison, N Chancellor, F Mintert & V Kendon (2019): *Finding spin glass ground states using quantum walks*. New J. Phys. 21, p. 123022, doi:10.1088/1367-2630/ab5ca2.

[9] N. Chancellor (2017): *Modernizing Quantum Annealing II: Genetic algorithms with the Inference Primitive Formalism*. Available at `https://arxiv.org/abs/1609.05875`. ArXiv:1609.05875.

[10] N. Chancellor (2017): *Modernizing Quantum Annealing using Local Searches*. New J. Phys. 19(2), p. 023024, doi:10.1088/1367-2630/aa59c4.

[11] Andrew M. Childs (2009): *Universal computation by quantum walk*. Phys. Rev. Lett. 102, p. 180501, doi:10.1103/PhysRevLett.102.180501.

[12] Andrew M. Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann & Daniel A. Spielman (2003): *Exponential algorithmic speedup by a quantum walk*. In: Proc. 35th Annual ACM Symposium on Theory of Computing (STOC 2003), Assoc. for Comp. Machinery, New York, pp. 59–68, doi:10.1145/780542.780552.

[13] Andrew M. Childs, Edward Farhi & John Preskill (2001): *Robustness of adiabatic quantum computation*. Physical Review A 65(1), p. 012322, doi:10.1103/PhysRevA.65.012322.

[14] Andrew M. Childs & Jeffrey Goldstone (2004): *Spatial search by quantum walk*. Physical Review A 70(2), p. 022314, doi:10.1103/PhysRevA.70.022314.

[15] Andrew M. Childs, David Gosset & Zak Webb (2013): *Universal computation by multi-particle quantum walk*. Science 339, pp. 791–794, doi:10.1126/science.1229957.

[16] Vicky Choi (2010): *Adiabatic quantum algorithms for the NP-complete Maximum-Weight Independent set, Exact Cover and 3SAT problems*. Available at `https://arxiv.org/abs/1004.2226`. ArXiv:1004.2226.

[17] A. Ben Dodds, Viv Kendon, Charles S. Adams & Nicholas Chancellor (2019): *Practical designs for permutation-symmetric problem Hamiltonians on hypercubes*. Phys. Rev. A 100, p. 032320, doi:10.1103/PhysRevA.100.032320.

[18] A Ekert & J Jozsa (1998): *Quantum algorithms: entanglementenhanced information processing*. Phil. Trans. Royal Soc. A 356, pp. 1769–82, doi:10.1098/rsta.1998.0248.

[19] E. Farhi, J. Goldstone, S. Gutmann & M. Sipser (2000): *Quantum Computation by Adiabatic Evolution*. Available at `http://arxiv.org/quant-ph/abs/0001106`. ArXiv:quant-ph/0001106.

[20] E Farhi & S Gutmann (1998): *Quantum computation and decison trees*. Phys. Rev. A 58, pp. 915–928, doi:10.1103/PhysRevA.58.915.

[21] A. B. Finilla, M. A. Gomez, C. Sebenik & J. D. Doll (1994): *Quantum annealing: A new method for minimizing multidimensional functions*. Chem. Phys. Lett. 219, p. 343, doi:10.1016/0009-2614(94)00117-0.

[22] Lucien Hardy (2001): *Quantum theory from five reasonable axioms*. Available at `http://arxiv.org/quant-ph/abs/0101012`. ArXiv:quant-ph/0101012.

[23] A. Hartwig, F. Daske & S. Kobe (1984): *A recursive branch-and-bound algorithm for the exact ground state of Ising spin-glass models*. Computer Physics Communications 32(2), pp. 133 – 138, doi:10.1016/0010-4655(84)90066-3.

[24] C. Horsman, S. Stepney, R. C. Wagner & V. Kendon (2014): *When does a Physical System Compute?* Proc. Roy. Soc. A 470(2169), p. 20140182, doi:10.1098/rspa.2014.0182.

[25] D Horsman, V Kendon, S Stepney & P Young (2017): *Abstraction and representation in living organisms: when does a biological system compute?* In Giovagnoli R Dodig-Crnkovic G, editor: Representation and Reality in Humans, Other Living Organisms and Intelligent Machines, Studies in Applied Philosophy, Epistemology and Rational Ethics 28, Springer, pp. 91–116, doi:10.1007/978-3-319-43784-2_6.

[26] Richard I. G. Hughes (1997): *Models and representation*. Philosophy of science 64, pp. S325–S336, doi:10.1086/392611.

[27] T. Kadowaki & H. Nishimori (1998): *Quantum annealing in the transverse Ising model*. Phys. Rev. E 58, p. 5355, doi:10.1103/PhysRevE.58.5355.

[28] Michal Karski, Leonid Forster, Jai-Min Choi, Andreas Steffen, Wolfgang Alt, Dieter Meschede & Artur Widera (2009): *Quantum Walk in Position Space with Single Optically Trapped Atoms*. Science 325(5937), pp. 174–177, doi:10.1126/science.1174436.

[29] Kempe, Kitaev & Regev (2004): *The Complexity of the Local Hamiltonian Problem*. In K. Lodaya & M. Mahajan, editors: *Proc. 24th FSTTCS*, *LNCS* 3328, Springer, pp. 372–383, doi:10.1007/978-3-540-30538-5_31.

[30] V. Kendon, A. Sebald, S. Stepney, M. Bechmann, P. Hines & R. C. Wagner (2011): *Heterotic computing*. In C.S. Calude, J. Kari, I. Petre & G. Rozenberg, editors: *Unconventional Computation, LNCS*, 6714, Springer, Berlin, Heidelberg, pp. 113–124, doi:10.1007/978-3-642-21341-0_16.

[31] V Kendon, A Siebald & S Stepney, editors (2015): *Heterotic computing: exploiting hybrid computational devices*. *Phil. Trans. Royal Soc. A* 373, Royal Society, London, UK, doi:10.1098/rsta.2015.0091.

[32] Viv Kendon (2020): *How to compute using quantum walks*, doi:10.24350/CIRM.V.19600203. CIRM. Audiovisual resource.

[33] Daniel Lidar (2019): *Arbitrary-time error suppression for Markovian adiabatic quantum computing using stabilizer subspace codes*. Available at `http://arxiv.org/abs/1904.12028`.

[34] Bas Lodewijks (2019): *Mapping NP-hard and NP-complete optimisation problems to Quadratic Unconstrained Binary Optimisation problems*. Available at `http://arxiv.org/abs/1911.08043`.

[35] N. B. Lovett, S. Cooper, M. S. Everitt, M. Trevers & V. Kendon (2010): *Universal quantum computation using the discrete time quantum walk*. *Phys. Rev. A* 81, p. 042330, doi:10.1103/PhysRevA.81.042330.

[36] M. Mohseni, P. Rebentrost, S. Lloyd & A. Aspuru-Guzik (2008): *Environment-assisted quantum walks in photosynthetic energy transfer*. *J. Chem. Phys.* 129, p. 174106, doi:10.1063/1.3002335.

[37] Ashley Montanaro (2018): *Quantum-Walk Speedup of Backtracking Algorithms*. *Theory of Computing* 14(15), pp. 1–24, doi:10.4086/toc.2018.v014a015.

[38] Ashley Montanaro (2019): *Quantum speedup of branch-and-bound algorithms*. Available at `http://arxiv.org/abs/1906.10375`. ArXiv:1906.10375.

[39] JG Morley, N Chancellor, S Bose & V Kendon (2019): *Quantum search with hybrid adiabatic-quantum walk algorithms and realistic noise*. *Phys. Rev. A* 99, p. 022339, doi:10.1103/PhysRevA.99.022339.

[40] Hagai B. Perets, Yoav Lahini, Francesca Pozzi, Marc Sorel, Roberto Morandotti & Yaron Silberberg (2008): *Realization of quantum walks with negligible decoherence in waveguide lattices*. *Phys. Rev. Lett.* 100, p. 170506, doi:10.1103/PhysRevLett.100.170506.

[41] Gualtiero Piccinini (2017): *Computation in Physical Systems*. In Edward N. Zalta, editor: *The Stanford Encyclopedia of Philosophy*, Summer 2017 edition, Stanford University Press. Available at `http://plato.stanford.edu/archives/sum2017/entries/computation-physicalsystems/`.

[42] Hilary Putnam (1988): *Representation and Reality*. MIT Press, Cambridge, MA. Available at `https://mitpress.mit.edu/books/representation-and-reality`. ISBN: 9780262161084.

[43] Jérémie Roland & Nicolas J. Cerf (2002): *Quantum search by local adiabatic evolution*. *Phys. Rev. A* 65, p. 042308, doi:10.1103/PhysRevA.65.042308.

[44] C. A. Ryan, M. Laforest, J. C. Boileau & R. Laflamme (2005): *Experimental implementation of discrete time quantum random walk on an NMR quantum information processor*. *Phys. Rev. A* 72, p. 062317, doi:10.1103/PhysRevA.72.062317.

[45] A. Schreiber, K. N. Cassemiro, V. Potoček, A. Gábris, I. Jex & Ch. Silberhorn (2011): *Decoherence and disorder in quantum walks: From ballistic spread to localization*. *Phys. Rev. Lett.* 106, p. 180403, doi:10.1103/PhysRevLett.106.180403.

[46] Neil Shenvi, Julia Kempe & K Birgitta Whaley (2003): *A quantum random walk search algorithm*. *Phys. Rev. A* 67, p. 052307, doi:10.1103/PhysRevA.67.052307.

[47] Steffen, vanDam, Hogg, Breyta & Chuang (2003): *Experimental implementation of an adiabatic quantum optimization algorithm*. *Phys. Rev. Lett.* 90(6), p. 067903, doi:10.1103/PhysRevLett.90.067903.

[48] S. Stepney & V. Kendon (2019): *The role of the representational entity in physical computing*. In: *UCNC 2019, Tokyo, Japan, June 2019*, *LNCS* 11493, Springer, pp. 219–231, doi:10.1007/978-3-030-19311-9_18.

[49] K. Wiesner (2009): *Quantum Cellular Automata*. In Robert A. Meyers, editor: *Springer Encyclopedia of Complexity and System Science*, chapter Cellular Automata, Mathematical Basis of, Ed. Andy Adamatzky, Springer, p. 00105, doi:10.1007/978-0-387-30440-3_426. or http://arxiv.org/abs/0808.0679.