Chapter 11
Computability and Complexity of Unconventional Computing Devices

Hajo Broersma, Susan Stepney, and Göran Wendin

Abstract We discuss some claims that certain UCOMP devices can perform hypercomputation (compute Turing-uncomputable functions) or perform super-Turing computation (solve $\mathcal{NP}$-complete problems in polynomial time). We discover that all these claims rely on the provision of one or more unphysical resources.

11.1 Introduction

For many decades, Moore's Law (Moore, 1965) gave us exponentially increasing classical (digital) computing (CCOMP) power, with a doubling time of around 18 months. This cannot continue indefinitely, due to ultimate physical limits (Lloyd, 2000). Well before then, more practical limits will slow this increase. One such limit is power consumption. With present efforts toward exascale computing, the cost of raw electrical power may eventually be the limit to the computational power of digital machines: Information is physical, and electrical power scales linearly with computational power (electrical power = number of bit flips per second times bit energy). Reducing the switching energy of a bit will alleviate the problem and push the limits to higher processing power, but the exponential scaling in time will win in the end. Programs that need exponential time will consequently need exponential electrical energy. Furthermore, there are problems that are worse than being hard for CCOMP: they are (classically at least) *undecidable* or *uncomputable*, that is, impossible to solve.

CCOMP distinguishes three classes of problems of increasing difficulty (Garey and Johnson, 1979):

1. Easy (tractable, feasible) problems: can be solved by a CCOMP machine, in polynomial time, $O(n^k)$, or better.

S. Stepney et al. (eds.), *Computational Matter*, Natural Computing Series,
https://doi.org/10.1007/978-3-319-65826-1_11

2. Hard (intractable, infeasible) problems: take at least exponential time, $O(e^n)$, or exponential resources like memory, on a CCOMP machine.
3. Impossible (undecidable, uncomputable) problems: cannot be solved by a CCOMP machine with any (finite) amount of time or memory resource.

Unconventional Computing (UCOMP) (European Commission, 2009) is a diverse field including a wealth of topics: hypercomputation, quantum computing (QCOMP), optical computing, analogue computing, chemical computing, reaction-diffusion systems, molecular computing, biocomputing, embodied computing, Avogadro-scale and amorphous computing, memcomputing, self-organising computers, and more.

One often hears that UCOMP paradigms can provide solutions that go beyond the capabilities of CCOMP (Konkoli and Wendin, 2014). There is a long-held notion that some forms of UCOMP can provide tractable solutions to $\mathcal{NP}$-hard problems that take exponential resources (time and/or memory) for CCOMP machines to solve (Adleman, 1994; Copeland, 2004; Lipton, 1995; Ouyang et al., 1997; Siegelmann, 1995), and the challenge to solve $\mathcal{NP}$-hard problems in polynomial time with finite resources is still actively explored (Manea and Mitrana, 2007; Traversa and Di Ventra, 2017; Traversa et al., 2015). Some go further, to propose UCOMP systems that can handle classically undecidable or uncomputable problems (Cabessa and Siegelmann, 2011; Copeland and Shagrir, 2011; Hogarth, 1992).

Many of these analyses may be *theoretically* sound, in that, if it were possible to implement the schemes, they would behave as claimed. But, *is* it possible to implement such schemes, to build such a computer in the material world, under the constraints of the laws of physics? Or are the hypothesised physical processes simply too hard, or impossible, to implement?

Key questions we discuss in this chapter are:

1. Can UCOMP provide solutions to classically undecidable problems?
2. Can UCOMP provide more effective solutions to $\mathcal{NP}$-complete and $\mathcal{NP}$-hard problems?
3. Are classical complexity classes and measures appropriate to any forms of UCOMP?
4. Which forms of UCOMP are clearly and easily amenable to characterisation and analysis by these? And why?
5. Are there forms of UCOMP where traditional complexity classes and measures are not appropriate, and what alternatives are then available?

The idea that Nature is physical and does not effectively solve $\mathcal{NP}$-hard problems does not seem to be generally recognised or accepted by the UCOMP community. However, there is most likely no free lunch with UCOMP systems providing shortcuts, actually solving $\mathcal{NP}$-hard problems (Aaronson, 2005). The question is then, what is the real computational power of UCOMP machines: are there UCOMP solutions providing significant polynomial speed-up and energy savings, or more cost-effective solutions beyond the practical capability of CCOMP high performance computing, or different kinds of solutions for embodied problems, or something else? This is the subject of the discussion in this chapter.

In Sect. 11.2 we discuss what it means to be a computational problem. In Sect. 11.4 we discuss UCOMP and hypercomputation (computability) claims. In Sect. 11.5 we recap the classical definitions of computational complexity. In Sect. 11.6 we discuss the power of various quantum computing approaches. In Sect. 11.7 we discuss UCOMP and super-Turing computation (complexity) claims, and the actual computational power of a variety of UCOMP paradigms.

11.2 Computational problems and problem solving

In the context of problem solving, the term complexity of a problem is used to indicate the difficulty of solving that particular problem, in many cases relative to the difficulty of solving other problems. Two questions that need to be answered first are: what do we mean in this context by *problem* and by *problem solving*?

11.2.1 Difficulty

Within the area of CCOMP, solving a particular problem means developing an algorithmic procedure that is able to produce a solution to that problem. This assumes that the problem consists of a set of instances, each of which can be encoded as an input to the algorithmic procedure, and that the algorithmic procedure then produces an output that can be decoded into a solution for that instance of the problem. This implies that being able to solve such types of problems means being able to write and install a computer program on a digital device that, executed on an input representing any instance of the problem produces an output that serves as a solution to that particular instance.

This leads to two natural questions:

- Does an algorithm exist for solving a particular problem? This is a question of *decidability* or *computability.*
- If such an algorithm does exist, how efficient is it at solving the problem? This is a question of *complexity.*

If such a computer program is available, it is natural to measure the difficulty of solving the problem by the *time* it takes the computer program to come up with the solution. There are many issues with this measure. For example, the execution time depends on the type and speed of the computer (processor), the type and size of the (encoding of the) instance, and on how smart the designed algorithmic procedure and its implementation were chosen.

In order to tackle some of these issues, the theory usually involves just the number of basic computational steps in the algorithmic procedure, and relates this to a function in the size of the instances. Upper bounds on the value of this function for the worst case instances are taken to indicate the relative complexity of the problem when compared to other problems.

Another natural question to ask is how much *space* (memory) does the program need to use to solve the problem. Again, the analyses abstract away from the complexities of actual computer memory (caches, RAM, discs, and so on), to an abstract concept of a unit of space.

This approach does not immediately say whether a more complex problem is intrinsically *difficult*, nor whether the algorithmic procedure used is optimal or not in terms of the complexity. Identification of the least complex algorithm for problems is at the heart of the theory of computational complexity.

11.2.2 Decision, optimisation, and counting problems

There are different types of problems. One distinction is based on the type of solutions.

The main focus in the area of computational complexity is on *decision problems*, where the solution for each problem instance is YES or NO. The task is, given an arbitrary instance and a certain fixed property, to answer whether the given instance has that property.

Consider the travelling salesman problem (TSP). An instance of TSP comprises a set of cities, together with the mutual distances between all pairs of cities. A route is a permutation of the city list, corresponding to travelling through each city precisely once, returning to the starting city. The length of the route is the sum of the distances between the cities as travelled. Given some value x representing the length of the route, TSP can be cast as a decision problem: is there a route that does not exceed x? For a nice exposition on the many facets of TSP we refer the reader to Lawler et al. (1985).

Consider the k-SAT (satisfiability) problem (Garey and Johnson, 1979). A formula (instance) in this problem involves any number m of conjoined clauses, each comprising the disjunction of k terms. Each clause's k terms are drawn from a total of n Boolean literals, $b_1 \dots b_n$, and their negations. For example, a 3-SAT problem instance could be the formula $(b_1 \vee b_2 \vee b_3) \wedge (\neg b_2 \vee b_3 \vee b_5) \wedge (b_1 \vee \neg b_3 \vee b_4) \wedge (\neg b_1 \vee b_3 \vee \neg b_5)$, which has $n = 5$ and $m = 4$. k-SAT is a decision problem: is there an assignment of truth values to the b_i that satisfies (makes true) the formula?

Decision problems differ from problems for which the solution is other than just YES or NO. A large class of problems for which this is the case, is the class of so-called *optimisation problems*. For these problems, it is not sufficient to come up with solutions, but the solutions are required to satisfy certain additional optimisation criteria. TSP can be cast as an optimisation problem: what is (the length of) a shortest route?

Another large class of problems that are not decision problems, is the class of *counting problems*. For these problems, the solutions are numbers rather than YES or NO. For TSP, one could, e.g., ask for the number of routes that are shorter than x, or for the number of different shortest routes.

Most optimisation and counting problems have decision counterparts (as is clear in the case of TSP above). Such optimisation and counting problems are obviously at least as difficult to solve as their decision counterparts.

11.2.3 Terminology

We use the term *hypercomputation* to refer to UCOMP models that can compute classically uncomputable functions (such as the Halting function, a total function that decides whether an arbitrary computer program halts on an arbitrary input). This is sometimes referred to as computation that "breaks the Turing barrier" or is "above the Turing limit" (that is, the barrier to, or limit on, computability).

We use the term *super-Turing computation* to refer to UCOMP models that can compute more efficiently (using exponentially fewer resources) than a Deterministic Turing Machine (DTM).

The UCOMP literature is not consistent in its use of these terms. Careful reading may be needed to determine if a particular claim is about computability or about complexity.

11.3 A brief review of CCOMP computability

11.3.1 Undecidable problems, uncomputable functions

Not all problems can be solved by an algorithmic procedure using a classical computer. It has been known since Turing (1937) that there are *undecidable* problems: those for which there is provably no algorithmic procedure to produce the correct YES/NO answer.

The earliest example of such a problem is the *Halting Problem*. In this problem, one has to write a computer program H that takes as its input any computer program P and input I, and outputs YES if P would eventually halt (terminate) when run on I, and outputs NO otherwise. There is provably no such H. Since then, many other examples of such undecidable problems have been established.

For problems not cast as decision problems, but in terms of computing the value of a function defined in a finite number of well-defined steps, there are *uncomputable* functions. Well-known examples include Kolmogorov complexity (Li and Vitányi, 1997), the Busy Beaver function (Rado, 1962), and Chaitin's omega halting probability (Chaitin, 1975; Chaitin, 2012). Note that by function in the above we mean a function on the natural numbers; such a function F is *(Turing) computable* if there is a Turing Machine that, on input n, halts and returns output $F(n)$. The use of Turing Machines here is not essential; there are many other models of computation that have the same computing power as Turing Machines.

The existence of (many) uncomputable functions of the above type follows from the fact that there are only *countably* many Turing Machines, and thus only countably many computable functions, but there are uncountably many functions on the natural numbers. Similarly, a set of natural numbers is said to be a *computable set* if there is a Turing Machine that, given a number n, halts with output 1 if n is in the set and halts with output 0 if n is not in the set. So for any set with an uncountable number of elements, most of its elements will be uncomputable. Hence most subsets of the natural numbers are uncomputable.

Decision problems can be encoded as subset problems: encode the problem instance as a unique natural number; the YES answers form a subset of these numbers; the decision problem becomes: is the number corresponding to the problem instance an element of the YES set? Hence most decision problems are uncomputable, that is, undecidable.

These undecidable problems and uncomputable functions are hard to solve or compute in a very strong sense: within the context of CCOMP it is simply impossible to solve or compute them.

11.3.2 Oracles and advice

Computability is an all or nothing property (although whether a problem class is computable may itself be an uncomputable problem). *Oracles* can be used to add nuance to this property: how much (uncomputable) help would be needed to make a problem computable? Less powerful oracles can also be considered when investigating complexity: how much oracular help is required to reduce the complexity of a problem class?

An oracle is an *abstract* black box that can take an input question from a DTM and output the answer. Oracles can be posited that provide answers to certain classes of problems, such as halting-problem oracles and $\mathcal{NP}$-problem oracles. An oracle is usually deemed to provide its answer in one step. (See Sect. 11.5.4 for a definition of classes $\mathcal{P}$ and $\mathcal{NP}$.)

Oracles can be posited, and their consequent abilities investigated theoretically, but they cannot be implemented on a classical computer, since they provide computational power above that of a DTM.

More recently introduced complexity classes try to capture additional computational power provided by allowing *advice* strings. An advice string is an extra input to a DTM that is allowed to depend on the length of the original input to the DTM, but not on the value of that input. A decision problem is in the complexity class $\mathcal{P}/f(n)$ if there is a DTM that solves the decision problem in polynomial time for any instance x of size n given an advice string of length $f(n)$ (not depending on x).

Trivially, any decision problem is in complexity class $\mathcal{P}/\text{exp}$. If the input is of size n, then there are $O(2^n)$ possible input values x of size n. An exponentially large advice string can enumerate the $O(2^n)$ YES/NO answers to the decision problem as an exponentially large lookup table.

Advice strings can be posited, and their consequent abilities investigated theoretically. Given an advice string, it can be implemented along with the DTM using its advice, since the string could be provided as an input to the DTM. However, classically, the advice on that string would itself have to be computed somehow; if the string contains uncomputable advice, then classically it cannot exist to be provided to the DTM.

11.3.3 Church–Turing thesis

The *Church–Turing Thesis* (CTT) states that "every number or function that 'would naturally be regarded as computable' can be calculated by a Turing Machine" (Copeland, 2015). This is a statement about computability, in terms of a (digital) classical computer.

Vergis et al. (1986) reformulate this thesis in terms of analogue computers as: "any analogue computer with finite resources can be simulated by a digital computer". This is a statement about computability: (finite) analogue computers do not increase what is computable over classical digital computers.

Hypercomputation seeks to discover approaches that can expand the range of computable functions beyond those computable by Turing Machines; it seeks to invalidate the CTT.

11.4 Hypercomputation

11.4.1 Undecidable problems determined physically?

Hypercomputation is a diverse field with many ideas on how to compute classically uncomputable functions using physical and non-physical approaches. One of the major proponents of hypercomputation is Jack Copeland (Copeland, 2004; Copeland, 2015; Copeland and Shagrir, 2011). Arkoudas (2008) states:

> Copeland and others have argued that the CTT has been widely misunderstood by philosophers and cognitive scientists. In particular, they have claimed that the CTT is in principle compatible with the existence of machines that compute functions above the "Turing limit", and that empirical investigation is needed to determine the "exact membership" of the set of functions that are physically computable.

Arkoudas (2008) disputes this argument, and claims that it is a category error to suggest that what is computable can be studied empirically as a branch of physics, because computation involves an *interpretation* or *representation* component, which is not a concept of the physical sciences. (See also Horsman et al. (2014).)

11.4.2 Accelerated Turing Machines

An example of a theoretical hypercomputer is the Zeno Machine (Potgieter, 2006). A Zeno Machine is an Accelerated Turing Machine that takes $1/2^n$ units of time to perform its n-th step; thus, the first step takes 1/2 units of time, the second takes 1/4, the third 1/8, and so on, so that after one unit of time, a countably infinite number of steps will have been performed. In this way, this machine formally solves the Halting Problem: is it halted at $t = 1$?.

Such a machine needs an exponentially growing bandwidth (energy spectrum) for operation, which is not a physically achievable resource.

Any physical component of such a machine would either run up against relativistic limits, and be moving too fast, or quantum limits, as it becomes very small, or both. The model implicitly relies on Newtonian physics.

11.4.3 General relativistic machines

There are various models that use General Relativistic effects to allow the computer to experience a different (and infinite) proper time from the (finite) time that the observer experiences. The best known of these is the Malament–Hogarth spacetime model (Etesi and Németi, 2002; Hogarth, 1992). The underlying concept is that the computer is thrown into one of these spacetimes, where it can be observed externally. If a computation does not halt, this can be determined in a finite time by the observer in the external reference frame, and so the set-up solves the Halting Problem.

This is an interesting branch of work, as it demonstrates clearly how the underlying laws of physics in the computer's material world can affect the reasoning used about possible computations.

However, there are several *practical* issues with this set-up. The computer has to be capable of running for an infinite time in its own reference frame. Also, the "tape" (memory) of the computer needs to have the potential to be actually infinite, not merely unbounded. It is not clear that such infinite time and infinite space can be physically realised.

11.4.4 Real number computation

A model of computation beyond the Turing limit has been formulated by Siegelmann (1995), involving neural networks with real-valued weights. Douglas (2003) and Douglas (2013) provides a critical analysis. The problem is the usual one for analogue systems: ultimate lack of precision; *in the end one needs exponential resources.* Analogue precision can be converted (by an ADC, Analogue-Digital Converter) into a corresponding digital range, which is effectively a memory requirement. $\mathcal{NP}$-problems (see later) require exponentially growing analogue precision, corresponding to a need for exponentially growing memory. Hypercomputational problems (computability) correspond to a need for infinite precision.

Real number hypercomputation (Blum, 2004) relies on physical systems being measurable to infinite precision. The underlying argument appears to be: physicists model the physical world using real numbers; real numbers have infinite precision and so contain infinite information; hence physical systems have infinite information; this information can be exploited to give hypercomputation. There are two problems with this argument.

The first problem is that the argument confuses the model and the modelled physical reality. Just because a quantity is modelled using a real number does not mean that the physical quantity faithfully implements those real numbers. The real-number model is in some sense 'richer' than the modelled reality; it is this extra richness that is being exploited in the theoretical models of hypercomputation. For example, consider Lotka–Volterra-style population models (Wangersky, 1978), where a real-valued variable is used to model the population number, which is in reality a discrete quantity: such models break down when the continuum approximation is no longer valid. Fluid dynamics has a continuous model, but in the physical world the fluid is made of particles, not a continuum, and so the model breaks down. The Banach–Tarski paradox (Wagon, 1985; Wapner, 2005) proves that it is possible to take a sphere, partition it into a finite number of pieces, and reassemble those pieces into two spheres each the same size as the original; the proof relies on properties of the reals that cannot be exploited to double a physical ball of material.

Secondly, even if some physical quantity were to contain arbitrary precision, there is strong evidence that it takes an exponentially increasing time to extract each further digit of information (see Sect. 11.5.11).

11.4.5 Using oracles, taking advice

Cabessa and Siegelmann (2011) state:

> The computational power of recurrent neural networks is intimately related to the nature of their synaptic weights. In particular, neural networks with static rational weights are known to be Turing equivalent, and recurrent networks with static real weights were proved to be [hypercomputational]. Here, we study the computational power of a more biologically-oriented model where the synaptic weights can evolve rather than stay static. We prove that such evolving networks gain a [hypercomputational] power, equivalent to that of static real-weighted networks, regardless of whether their synaptic weights are rational or real. These results suggest that evolution might play a crucial role in the computational capabilities of neural networks.

A proposed rational-number hypercomputer avoids the issue of infinite precision. The set-up described by Cabessa and Siegelmann (2011) is a neural network with rational, but changing ('evolving'), weights. However, the changing weights are not computed by the network itself, nor are they provided by any kind of evolutionary feedback with a complex environment. They are provided directly as input in the form of a sequence of increasing-precision rational numbers: that is, an advice string.

Any claim of hypercomputation achieved through the use of an oracle or advice needs to address the feasibility of implementing said oracle or advice. These can provide hypercomputational power only if they are themselves Turing-uncomputable,

11.4.6 Conclusion

Hypercomputation models tend to rely on one or more of:

- known incorrect models of physics (usually Newtonian, ignoring relativistic and/or quantum effects)
- physically-instantiated infinities (in time and/or space and/or some other physical resource)
- physically accessible infinite precision from real numbers
- Turing-uncomputable oracles or advice

There is currently no evidence that any of these essentially *mathematical* hypercomputation models are physically realisable. Their study is interesting, however, because they illuminate the various relationships between computability and physical (as opposed to mathematical) constraints. For example, they bring into focus an unconventional computational resource: precision.

Other facets of hypercomputation, of moving beyond computational paradigms other than Turing, are discussed in Stepney (2009).

11.5 A brief review of CCOMP complexity

We now move on to discussing super-Turing UCOMP models, which deal with computational complexity. First we briefly review some concepts of classical complexity theory. Further information can be found in any good textbook on computational complexity, such as Garey and Johnson (1979) and Sipser (1997).

11.5.1 Measuring complexity

Complexity in the classical setting of digital computing is typically a mathematically calculated or proven property, rather than an empirically measured property, for two main reasons.

Firstly, complexity refers to asymptotic properties, as the problem sizes grow. It would have to be measurable over arbitrarily large problem sizes to determine its asymptotic behaviour. This is particularly challenging for many UCOMP devices (including quantum computers) that to date only exist as small prototypes that can handle only small problem instances.

Secondly, complexity is a worst case property: the complexity of a problem (or class) is the complexity of the hardest problem instance (or hardest problem in that class). Some instances can be easy, other instances hard. If there are very few hard instances, these "pathological" instances may not be encountered during empirical sampling.

11.5.2 Easy, polynomial time problems

Edmonds (1965) was the first to distinguish good and bad algorithmic procedures for solving decidable problems. He coined the term *good algorithm* for an algorithmic procedure that produces the correct solution using a number of basic computational steps that is bounded from above by a polynomial function in the instance size. He also conjectured that there are decidable problems for which such good algorithms cannot be designed. This conjecture is still open, although there is a lot of evidence for its validity. We come back to this later.

The algorithms that Edmonds called good, are nowadays usually referred to as *polynomial* (time) algorithms, or algorithms with a polynomial (time) complexity. The corresponding problems are usually called polynomial(ly solvable) problems, but also referred to as easy, tractable, or feasible problems.

Define the function $f(n)$ to be the bound on an algorithm's number of basic computational steps in the worst case, for an instance of size n. Then for a polynomial (time) algorithm, $f(n)$ is $O(n^k)$, meaning that there exists a positive integer k and positive constants c and n_0 such that $f(n) \leq c \cdot n^k$ for all $n \geq n_0$. The n_0 is included in the definition because small problem instances do not determine the complexity: the complexity is characterised by what happens for *large* problem instances.

11.5.3 Hard, exponential time problems

Decidable decision problems for which no tractable (polynomial time) algorithm exists are called hard, intractable, or infeasible problems. These usually allow straightforward exponential algorithms: algorithmic procedures for solving them have worst case instances that take an exponential number of computational steps. The function $f(n)$ that bounds the number of computational steps, in the worst case for an instance of size n, is $O(c^n)$, for some positive constant c.

Similar to the conjecture of Edmonds (1965), it is nowadays widely believed that there are decision problems for which the only possible algorithmic procedures for solving them have an exponential complexity.

11.5.4 Complexity classes $\mathcal{P}$ and $\mathcal{NP}$

Based on the above distinction between easy and hard problems, the formally defined complexity class $\mathcal{P}$ consists of all decision problems that are tractable, that admit polynomial algorithms for solving them by a *classical Deterministic Turing Machine (DTM)*. A decision problem in $\mathcal{P}$ is also referred to as a problem with a polynomial complexity, or simply as a polynomial problem. The decision version of TSP described above has no known polynomial time algorithm to solve it.

Many textbooks, such as Garey and Johnson (1979) and Sipser (1997), provide a fundamental treatment of complexity class $\mathcal{P}$ in terms of Turing

Machines. There, it is argued that the problems in $\mathcal{P}$ are precisely those problems that can be encoded and decided on a DTM within a number of transitions between states that is bounded by a polynomial function in the size of the encoding (number of symbols) of the input.

The class $\mathcal{NP}$ can be defined in terms of Turing Machines as consisting of those problems that can be decided on a *Non-deterministic Turing Machine (NTM)* in polynomial time. A DTM has only one possible move at each step, determined by its state transition function along with its internal and tape state. In contrast, an NTM has potentially several alternative moves available at each step, and chooses one of these non-deterministically. The computation succeeds if at least one sequence of possible choices succeeds.

There are alternative ways to consider the working of an NTM: (i) it uses *angelic non-determinism* and always makes the correct choice; (ii) at each choice point, it 'branches' into parallel machines, taking all possible paths (hence using exponential space). An NTM can be implemented by serialising this branching approach (Floyd, 1967): if it has chosen a path and discovers it is the wrong path, it 'backtracks' and makes a different choice (hence potentially using exponential time). Hence a DTM can compute anything an NTM can compute, although potentially exponentially slower.

An NTM can also be considered as an *oracle machine*, a black box that provides candidate solutions for a specific class of problems (see Sect. 11.3.2). In the terminology used in textbooks like Garey and Johnson (1979) and Sipser (1997), a decision problem is in the class $\mathcal{NP}$ if, for any YES-instance of the problem there is a candidate solution that can be checked by an algorithmic procedure in polynomial time. So, instead of finding the correct answer for any instance in polynomial time, it is only required to be able to verify the correctness of a candidate solution for the YES-answer for any YES-instance in polynomial time.

The decision version of TSP described above is in $\mathcal{NP}$: the relevant candidate solution is a suitable short route, and the length of that given route can be calculated in polynomial time and checked to be at most x. Interpretation (i) of the NTM above has it 'angelically' making the correct choice at each city; interpretation (ii) has it exploring all exponential number of possible paths in parallel.

It is clear that $\mathcal{P} \subseteq \mathcal{NP}$: if a problem can be solved in polynomial time, it can certainly be checked in polynomial time.

It is widely believed that $\mathcal{P} \neq \mathcal{NP}$ (and that is the position we take in this chapter). Its proof (or disproof) is a fundamental open problem within mathematics and theoretical computer science (Aaronson, 2017). Many decision problems have been shown to be in $\mathcal{P}$, but for even more, such as TSP, it is not known whether they are in $\mathcal{P}$ or not.

11.5.5 $\mathcal{NP}$-complete problems

There are many decision problems for which the complexity status is currently unknown. To say at least something about their relative complexity, Cook (1971) and Levin (1973) developed useful machinery, which has led to the definition of the class of $\mathcal{NP}$-complete problems.

A problem in $\mathcal{NP}$ is called *$\mathcal{NP}$-complete* if it is the hardest of all problems in $\mathcal{NP}$, in the following sense. Consider two problems P and Q that are both in $\mathcal{NP}$. Suppose that there exists a polynomial reduction from P to Q, that is, a polynomial algorithm to transform any instance I of P into an instance J (of size bounded by a polynomial function in the size of I) of Q in such a way that I is a YES-instance of P if and only if J is a YES-instance of Q. Then any polynomial algorithm for solving Q can be transformed into a polynomial algorithm for solving P. In the sense of polynomial complexity, in such a case Q is at least as hard to solve as P. If the same holds for Q and any other problem instead of P in $\mathcal{NP}$, then Q is the hardest of all problems in $\mathcal{NP}$, in the above sense. Such a problem Q in $\mathcal{NP}$ is an $\mathcal{NP}$-complete problem.

Cook (1971) and Levin (1973) independently showed that there are $\mathcal{NP}$-complete problems. They each proved that the unrestricted Boolean satisfiability problem (SAT) is $\mathcal{NP}$-complete. This was a major breakthrough, because it allowed many other problems to be shown to be $\mathcal{NP}$-complete, by using a polynomial reduction from a known $\mathcal{NP}$-complete problem (starting with SAT) to the newly considered problem in $\mathcal{NP}$ (Karp, 1972). The TSP and k-SAT (with $k \geq 3$) decision problems (Sect. 11.2.2) are both $\mathcal{NP}$-complete; the 2-SAT problem is in $\mathcal{P}$.

If a polynomial algorithm exists for any of these $\mathcal{NP}$-complete problems, then a polynomial algorithm would exist for each of them, by using the reduction process used in their proofs of $\mathcal{NP}$-completeness. The existence of an ever growing number of $\mathcal{NP}$-complete problems for which nobody to date has been able to develop a polynomial algorithm provides significant evidence (although not proof) supporting the conjecture $\mathcal{P} \neq \mathcal{NP}$.

11.5.6 $\mathcal{NP}$-hard problems

For problems other than decision problems, such as the optimisation and counting problems mentioned earlier, their computational complexity is usually defined only if they are in $\mathcal{NP}$ and contain decision problems in $\mathcal{NP}$ as special cases, and hence are at least as difficult to solve as their decision counterparts. Such problems are called *$\mathcal{NP}$-hard* if they are at least as hard as an $\mathcal{NP}$-complete problem, that is, if a polynomial time algorithm for solving them would imply a polynomial algorithm for solving an $\mathcal{NP}$-complete (decision) problem. For a compendium of $\mathcal{NP}$-hard optimisation problems, see Crescenzi and Kann (2005).

11.5.7 Other classic complexity classes: $\mathcal{PSPACE}$ and $\mathcal{BPP}$

Since the introduction of the $\mathcal{P}$ and $\mathcal{NP}$ complexity classes, a whole zoo of further complexity classes has been defined and studied. Most of these classes are beyond the scope of this chapter, but we mention a few here that are relevant in the context of this chapter.

The complexity class $\mathcal{PSPACE}$ consists of decisions problems that can be solved using polynomial space on a DTM, meaning that the number of cells on the tape of the DTM that are needed to encode and solve a given instance is bounded by a polynomial function in the length of the input size. Note that no constraint is put on the time allowed for the solution (other than being finite). For this class, using an NTM does not add any extra computational power in terms of space use, because an NTM that uses polynomial space can be simulated by a DTM that uses (more but still) polynomial space (but it may use substantially more time).

We clearly have $\mathcal{NP} \subseteq \mathcal{PSPACE}$: if a problem can be checked in polynomial time, it cannot use more than polynomial space, since it has to visit all of that space. It is widely believed that $\mathcal{NP} \neq \mathcal{PSPACE}$, but again, there is no proof.

The complexity class $\mathcal{BPP}$ (Bounded-error Probabilistic Polynomial) consists of decision problems that can be solved in polynomial time by a *probabilistic* Turing Machine (PTM), i.e., a DTM that can make random choices between different transitions according to some probability distribution. (This is distinct from an NTM: a probabilistic TM makes a *random* choice; an NTM makes the 'correct' choice, or all choices, depending on the interpretation.) The probability that any run of the algorithm gives the wrong answer to a YES-NO question must be less than 1/3.

It is obvious that $\mathcal{P} \subseteq \mathcal{BPP}$: if a problem can be solved in polynomial time, it can be probabilistically solved in polynomial time. In this case, it is widely believed that $\mathcal{P} = \mathcal{BPP}$, but yet again, there is no proof. There is no known subset relation between $\mathcal{BPP}$ and $\mathcal{NP}$, in either direction.

11.5.8 Quantum complexity classes

A *quantum* TM (QTM), with a quantum processor and quantum tape (memory) is a model for a quantum computer, computing directly in memory (Deutsch, 1985).

Problems that can be solved by a QTM in polynomial time belong to the complexity class $\mathcal{BQP}$ (Fig. 11.1) (*Complexity Zoo website* n.d.; Montanaro, 2016; Watrous, n.d.). $\mathcal{BQP}$ is in some sense the quantum analogue of the classical $\mathcal{BPP}$, but there is a fundamental difference: the PTM proceeds via

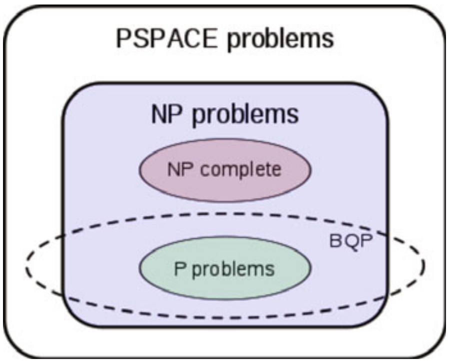

Fig. 11.1 Summary of relationships between computational complexity classes (https://en.wikipedia.org/wiki/BQP). See text for details.

random choices of unique states of the Finite State Machine reading and writing on the tape, while the QTM proceeds via quantum simultaneous superposition and entanglement of all the states. Therefore, the QTM proceeds through the Hilbert state space in a *deterministic* way via the time evolution operator defined by the Hamiltonian. The probabilistic aspects emerge when reading out the results; in some cases this can be done deterministically, in other cases one has to collect statistics.

Fig. 11.1 shows that $\mathcal{BQP}$ is a limited region of the complexity map, and (probably) does not include the $\mathcal{NP}$-complete class. There, in $\mathcal{BQP}$ and (probably) outside $\mathcal{P}$, we find problems like Shor's factorisation algorithm (Shor, 1997), providing exponential speed-up over the best known classical factorisation algorithm. Classic factorisation is believed to be neither $\mathcal{NP}$-complete, nor in $\mathcal{P}$. Another example is unstructured database search, which is classically "easy" (polynomial), but which shows quadratic speed-up with the Grover quantum search algorithm (Grover, 1996). See Montanaro (2016) for a recent overview of progress in the field, focusing on algorithms with clear applications and rigorous performance bounds.

There are further quantum complexity classes. In particular, $\mathcal{QMA}$ is the class where problems proposed by a quantum oracle can be verified in polynomial time by a quantum computer in $\mathcal{BQP}$ (*Complexity Zoo website* n.d.; Montanaro, 2016; Watrous, n.d.).

11.5.9 Complexity with advice: $\mathcal{P}$/poly and $\mathcal{P}$/log

The most common complexity class involving advice is $\mathcal{P}$/poly, where the advice length $f(n)$ can be any polynomial in n. This class $\mathcal{P}$/poly is equal to the class consisting of decision problems for which there exists a polynomial size Boolean circuit correctly deciding the problem for all inputs of length

n, for every n. This is true because a DTM can be designed that interprets the advice string as a description of the Boolean circuit, and conversely, a (polynomial) DTM can be simulated by a (polynomial) Boolean circuit.

Interestingly, $\mathcal{P}$/poly contains both $\mathcal{P}$ and $\mathcal{BPP}$ and it also contains some undecidable problems (including the *unary* version of the Halting Problem). It is widely believed that $\mathcal{NP}$ is not contained in $\mathcal{P}$/poly, but again there is no proof for this. If has been shown that $\mathcal{NP} \not\subset \mathcal{P}$/poly implies $\mathcal{P} \neq \mathcal{NP}$. Much of the efforts towards proving that $\mathcal{P} \neq \mathcal{NP}$ are based on this implication.

The class $\mathcal{P}$/log is similar to $\mathcal{P}$/poly, except that the advice string for inputs of size n is restricted to have length at most logarithmic in n, rather than polynomial in n. It is known that $\mathcal{NP} \subseteq \mathcal{P}$/log implies $\mathcal{P} = \mathcal{NP}$.

Restricting the advice length to at most a logarithmic function of the input size implies that polynomial reductions cannot be used to show that a decision problem belongs to the class $\mathcal{P}$/log. To circumvent this drawback the prefix advice class Full-$\mathcal{P}$/log has been introduced (Balcázar and Hermo, 1998). The difference with $\mathcal{P}$/log is that in Full-$\mathcal{P}$/log each advice string for inputs of size n can also be used for inputs of a smaller size. Full-$\mathcal{P}$/log is also known as $\mathcal{P}/\log^*$ in the literature.

11.5.10 Extended Church–Turing thesis

The CTT (Sect. 11.3.3) is a statement about *computability.* The *Extended Church–Turing Thesis* (ECT) is a statement about *complexity*: any function naturally to be regarded as *efficiently* computable is efficiently computable by a DTM (Dershowitz and Falkovich, 2012). Here "efficiently" means computable by a DTM in polynomial time and space. A DTM is a basic model for ordinary *classical* digital computers solving problems tractable in polynomial time.

Consider the NTM (Sect. 11.5.4). If backtracking is the most efficient way to implement an NTM with a DTM (that is, if $\mathcal{P} \neq \mathcal{NP}$), then the ECT claims that a 'true' NTM cannot be implemented.

Vergis et al. (1986) reformulate the ECT in terms of analogue computers as: "any finite analogue computer can be simulated *efficiently* by a digital computer, in the sense that the time required by the digital computer to simulate the analogue computer is bounded by a polynomial function of the resources used by the analogue computer". That is, finite analogue computers do not make infeasible problems feasible. Thus, finite analogue computers cannot tractably solve $\mathcal{NP}$-complete problems.

Super-Turing computation seeks to discover approaches that can expand the range of efficiently computable functions beyond those efficiently computable by DTMs; it seeks to invalidate the ECT.

Quantum computing (QCOMP) can provide exponential speed-up for a few classes of problems (see Sect. 11.5.8), so the ECT is believed to have been invalidated in this case (Aaronson, 2013a; Aaronson, 2013b): quantum computing can provide certain classes of super-Turing power (unless $\mathcal{P} = \mathcal{NP}$). We can extend the ECT to: "any function naturally to be regarded as *efficiently* computable is efficiently computable by a Quantum Turing Machine (QTM)", and then ask if any *other* form of UCOMP can invalidate either the original ECT, or this quantum form of the ECT.

11.5.11 Physical oracles and advice

An interesting question in UCOMP is whether a *physical* system can be implemented that acts as some specific oracle or advice, that is, whether it is possible to build a physical add-on to a classical computer that can change the computability or complexity classes of problems.

A range of analogue devices have been posited as potential physical oracles. Their analogue physical values may be (theoretically) read with infinite, unbounded, or fixed precision, resulting in different (theoretical) oracular power.

Beggs and coauthors (see Ambaram et al. (2017) and references therein) have made a careful analysis of using a range of idealised physical experiments as oracles, in particular, studying the time it takes to interact with the physical device, as a function of the precision of its output. More precision takes more time. In each system they analyse, they find that the time needed to extract the measured value of the analogue system increases exponentially with the number of bits of precision of the measurement. They conjecture that

> for all "reasonable" physical theories and for all measurements based on them, the physical time of the experiment is at least exponential, i.e., the time needed to access the n-th bit of the parameter being measured is at least exponential in n.

The kind of physical analogue devices that Ambaram et al. (2017) analyse tend to use a *unary* encoding of the relevant value being accessed via physical measurement, for example, position, or mass, or concentration. So each extra digit of precision has to access an exponentially smaller range of the system being measured. See Fig. 11.2.

Similar points apply when discussing the input size n when analysing such devices: if the UCOMP device uses a unary encoding of the relevant parameter values, as many do, the input size n is exponentially larger than if a binary encoding were used.

Beggs et al. (2014) use such arguments to derive an *upper bound* on the power of such hybrid analogue-digital machines, and conjecture an associated "Analogue-digital Church–Turing Thesis":

> No possible abstract analogue-digital device can have more computational capabilities in polynomial time than $\mathcal{BPP}//\log^*$.

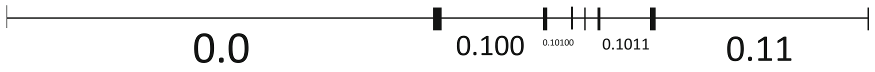

Fig. 11.2 Encoding a real number value as a position on a line. Extraction of each extra digit of precision requires access to an exponentially smaller region of the line.

Note that this conjecture refers to abstract (or idealised) physical devices, analogous to the way a DTM is also an abstract idealised device. Physical issues such as thermodynamic jitter and quantum uncertainty have still to be considered. Note also that the logarithmic advice has to be encoded somehow into the analogue device.

Blakey (2014) and Blakey (2017) has made a careful analysis of “unconventional resources” such as precision (measurement and manufacturing), energy, and construction material, and has developed a form of “model-independent” complexity analysis, that considers such unconventional resources in addition to the classical ones of space and time. Indeed, Blakey’s analysis shows that the exponential cost of measurement precision is itself outweighed by the *infinite* manufacturing precision required for certain devices. Blakey’s approach to analysing unconventional resources enables a formalisation of “the intuition that the purported [hypercomputational] power of these computers in fact vanishes once precision is properly considered”.

11.5.12 Complexity as a worst case property

$\mathcal{NP}$-completeness is a worst case analysis: a problem is $\mathcal{NP}$-complete if it has at least *one* instance that requires exponential, rather than polynomial, computation time, even if all the remaining instances can be solved in polynomial time.

Cheeseman et al. (1991) note that for many $\mathcal{NP}$-complete problems, typical cases are often easy to solve, and hard cases are rare. They show that such problems have an “order parameter”, and that the hard problems occur at the critical value of this parameter. Consider 3-SAT (Sect. 11.2.2); the order parameter is the average number of constraints (clauses) per Boolean literal, m/n. For low values the problem is underconstrained (not many clauses compared to literals, so easily shown to be satisfiable) and for high values it is overconstrained (many clauses compared to literals, so easily shown to be unsatisfiable). Only near a critical value do the problems become exponentially hard to determine.

Such arguments demonstrate that we cannot ‘sample’ the problem space to demonstrate problem hardness; complexity is not an experimental property. In particular, demonstrating that a device or process, engineered or natural, can solve some (or even many) $\mathcal{NP}$-complete problem instances tractably is not sufficient to conclude that it can solve all $\mathcal{NP}$-complete problem instances tractably.

The limitations that $\mathcal{NP}$-completeness imposes on computation probably hold for all natural analogue systems, such as protein folding, the human brain, etc. (Bryngelson et al., 1995). As noted above, just because Nature can efficiently solve *some* instances of problems that are $\mathcal{NP}$-complete does not mean that it can solve *all* $\mathcal{NP}$-complete problem instances (Bryngelson et al., 1995). To find the lowest free energy state of a general macromolecule has been shown to be $\mathcal{NP}$-complete (Unger and Moult, 1993). In the case of proteins there are amino acid sequences that cannot be folded to their global free energy minimum in polynomial time either by computers or by Nature. Proteins selected by Nature and evolution will represent a *tractable subset* of all possible amino acid sequences.

11.5.13 Solving hard problems in practice

Apart from trying to show that $\mathcal{P} = \mathcal{NP}$, there are other seemingly more practical ways to try to cope with $\mathcal{NP}$-complete or $\mathcal{NP}$-hard problems.

If large instances have to be solved, one approach is to look for fast algorithms, called *heuristics*, that give reasonable solutions in many cases. In some cases there are approximation algorithms for optimisation problems with provable approximation guarantees. This holds for the optimisation variant of the TSP restricted to instances for which the triangle equality holds (the weight of edge uv is at most the sum of the weights of the edges uw and wv, for all distinct triples of vertices u, v, w), and where one asks for (the length of) a shortest tour. This variant is known to be $\mathcal{NP}$-hard, but simple polynomial time heuristics have been developed that yield solutions within a factor of 1.5 of the optimal tour length (Lawler et al., 1985).

For many optimisation problems even guaranteeing certain approximation bounds is an $\mathcal{NP}$-hard problem in itself. This also holds for the general TSP (without the triangle inequality constraints) if one wants to find a solution within a fixed constant factor of the optimal tour length (Lawler et al., 1985).

A more recent approach tries to capture the exponential growth of solution algorithms in terms of a function of a certain fixed parameter that is not the size of the input. The aim is to develop a solution algorithm that is polynomial in the size of the input but maybe exponential in the other parameter. For small values of the fixed parameter the problem instances are tractable, hence the term fixed parameter tractability (the class $\mathcal{FPT}$) for such problems (Downey and Fellows, 1999).

An example is k-SAT (Sect. 11.2.2), parameterised by the number n of Boolean literals. A given formula of size N with n literals can be checked by brute force in time $O(2^n N)$, so linear in the size of the instance.

A related concept is that of preprocessing (data reduction or kernelisation). Preprocessing in this context means reducing the input size of the problem instances to something smaller, usually by applying reduction rules that take care of easy parts of the instances. Within parameterised complexity theory, the smaller inputs are referred to as the kernel. The goal is to prove that small kernels for certain $\mathcal{NP}$-complete or $\mathcal{NP}$-hard problems exist, and can be found in polynomial time. If small here means bounded by a function that only depends on some fixed parameter associated with the problem, then this implies that the problem is fixed parameter tractable.

The above definitions focus on worst case instances of the (decision) problems. It is not clear whether this is always a practical focus. There is a famous example of a class of problems in $\mathcal{P}$ – Linear Programming – for which empirical evidence shows that an exponential algorithm (the Simplex Method) for solving these problems very often yields faster solutions in practice than the polynomial algorithm (the Ellipsoid Method) developed subsequently (Papadimitriou, 1994).

For many algorithms, a worst case analysis gives limited insight into their performance, and can be far too pessimistic to reflect the actual performance on realistic instances. Recent approaches to develop a more realistic and robust model for the analysis of the performance of algorithms include average case analysis, smoothed analysis, and semi-random input models. All of these approaches are based on considering instances that are to a certain extent randomly chosen.

11.5.14 No Free Lunch theorem

Wolpert and Macready (1997) prove "no free lunch" (NFL) theorems related to the efficiency of search algorithms. They show that when the performance of any given search algorithm is averaged over all possible search landscapes, it performs no better than random search. This is because, whatever algorithm is chosen, if it exploits the structure of the landscape, there are always deceptive search landscapes that lead it astray. The only way not to be deceived is to search randomly. A problem of size n has a search space of size $O(2^n)$, and so the best classical search algorithm, where the performance is averaged over all the $O(2^{2^n})$ possible landscapes, is $O(2^n)$.

This does not mean that there are no search algorithms better than random search over certain subsets of search landscapes: algorithms that exploit any structure common across the subset can perform better than random. More generally, if some search landscapes are more likely than others, algorithms that can exploit that information can do better (Wolpert, 2012).

Natural processes such as Darwinian evolution, which may be interpreted as a form of search algorithm, are almost certainly exploiting the structure of their search landscapes. This has consequences for nature-inspired search algorithms, such as evolutionary algorithms, if they are to be exploited on 'unnatural' landscapes. See also the comments on protein folding in Sect. 11.5.12.

11.6 Quantum information processing

Quantum computers are able to solve some problems much faster than classical computers (*Complexity Zoo website* n.d.; Montanaro, 2016; Shor, 1997; Watrous, n.d.). However, this does not say much about solving computational problems that are hard for classical computers. If one looks at the map of computational complexity (Fig. 11.1), classifying the hardness of computational (decision) problems, one finds that the $\mathcal{BQP}$ class of quantum computation covers a rather limited space, not containing really hard problems. One may then ask what is the fundamental difference between CCOMP and QCOMP, and what kind of problems are hard even for a quantum computer (Aaronson, 2005; Aaronson, 2008; Aaronson, 2009; Aaronson, 2013b)?

11.6.1 Digital quantum computation

The obvious difference between CCOMP and QCOMP is that CCOMP is based on classical Newtonian physics and special and general relativity, while QCOMP is based on quantum physics, as illustrated in Figs. 11.3a,b. Digital CCOMP progresses by gate-driven transitions between specific classical memory configurations of an N-bit register $R(t_k)$, each representing one out of 2^N instantaneous configurations.

QCOMP, on the other hand, progresses by gate-driven transitions between specific quantum memory states $|\Psi(t_k)\rangle$, each representing instantaneous superposition of 2^N configurations. The quantum memory states are coherent amplitudes with well-defined phase relations. Moreover, the states of the qubits can be entangled, i.e., not possible to write as a product of states. In the case of two qubits, the canonical example of entanglement is that of Bell states: non-entangled product states are $|00\rangle$ or $|11\rangle$ or $(|0\rangle + |1\rangle)(|0\rangle + |1\rangle) = |00\rangle + |01\rangle + |10\rangle + |11\rangle$. The Bell states $|00\rangle \pm |11\rangle$ and $|0\rangle \pm |10\rangle$ are clearly not product states, and represent in fact maximum entanglement of two qubits. This can be generalised to more qubits, e.g., the Greenberger–Horne–Zeilinger (GHZ) "cat state": $|000\rangle + |111\rangle$. This entanglement represents non-classical correlations, at the heart of the exponential power of QCOMP. Entanglement allows us to construct maximally entangled superpositions with only a linear amount of physical resources, e.g., a large cat state: $\frac{1}{\sqrt{2}}(|0......00\rangle + |1.....11\rangle)$. This is what allows us to perform non-classical tasks and provide quantum speed-up (Horodecki et al., 2009; Jozsa and Linden, 2003).

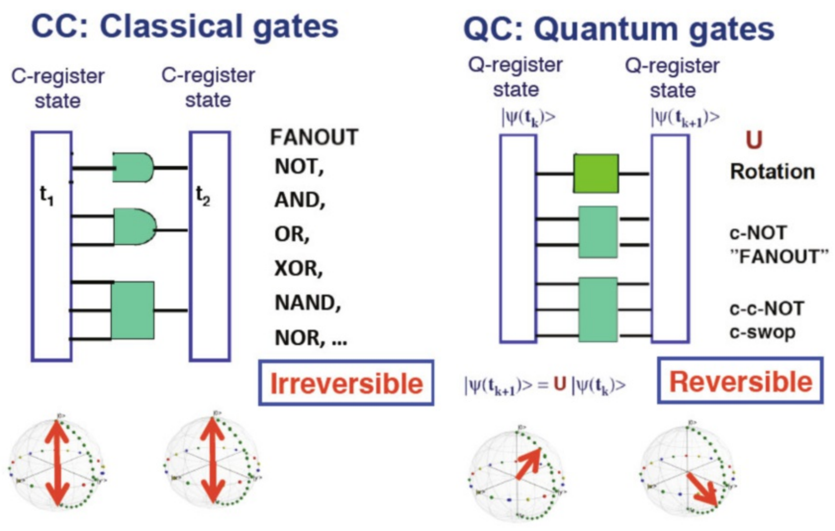

Fig. 11.3 a. Comparison of CCOMP and QCOMP. (left) CCOMP: irreversible gates with arithmetic-logic unit (ALU) and memory separated. The memory is the storage, with classical bits 0,1 representing the poles on the Bloch unit sphere. Classical gates are basically hardwired, irreversible and performed in the ALU. Gates are clocked. (right) QCOMP: Computing in memory – the memory is the computer. Quantum bits (qubits) $\alpha|0\rangle + \beta|1\rangle$ span the entire Bloch sphere. Quantum gates are reversible and performed on the "memory" qubits by software-controlled external devices. Gates are not clocked.

QCOMP is basically performed directly in memory. One can regard the qubit memory register as an array of 2-level quantum transistors, memory cells, where the gates driving and coupling the transistors are external *classical* fields controlled by classical software run on a CCOMP. This emphasises that a quantum computer can only implement a polynomial number of gates, and that the name of the game is to devise *efficient* decompositions of the time-evolution operator in terms of universal gates. The goal is of course to construct single-shot multi-qubit gates implementing long sequences of canonical elementary gates to synthesise the full time-evolution operator $U = exp(-iHt)$ (Fig. 11.3).

QCOMP depends, just like CCOMP, on encoding/decoding, error correction, and precision of measurement and control. To go beyond $\mathcal{BQP}$ essentially takes non-physical oracle resources, or unlimited precision, requiring exponential resources and ending up in $\mathcal{QMA}$ or beyond.

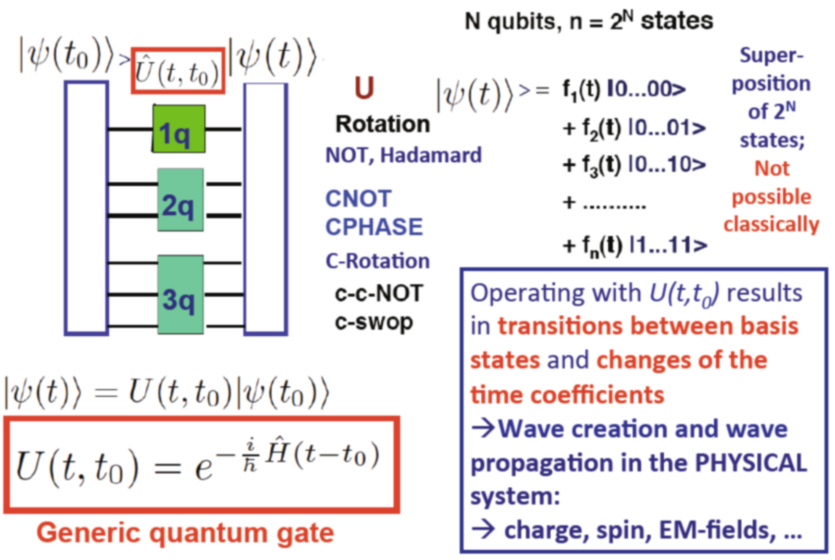
N qubits, n = 2^N states
$|\psi(t_0)\rangle$
$\hat{U}(t,t_0)$
$|\psi(t)\rangle$
1q
2q
3q
U
Rotation
NOT, Hadamard
CNOT
CPHASE
C-Rotation
c-c-NOT
c-swop
$|\psi(t)\rangle$ = f1(t) I0...00> + f2(t) I0...01> + f3(t) I0...10> + ... + fn(t) I1...11>
Super-position of 2^N states; Not possible classically
$|\psi(t)\rangle = U(t,t_0)|\psi(t_0)\rangle$
$U(t,t_0) = e^{-\frac{i}{\hbar}\hat{H}(t-t_0)}$
Generic quantum gate
Operating with U(t,t0) results in transitions between basis states and changes of the time coefficients
→Wave creation and wave propagation in the PHYSICAL system:
→ charge, spin, EM-fields, ...

Fig. 11.3 b. Comparison of CCOMP and QCOMP (ctd). (left) The time evolution of the state of the quantum computer is implemented by the unitary time evolution operator U, which can be broken down into elementary gates (like NOT, Hadamard, CNOT, C-Rotation). (right) The quantum state is, in general, an entangled superposition of all configurations of the memory register. Entanglement implies that the state is not a product state, containing non-classical correlations that provide polytime solution of certain problems that take exponential time for classical computers. The quantum gates are operated by a classical computer, which means that a quantum computer can only solve problems that take at most a polynomial number of gates to solve. A universal set of quantum gates (Hadamard (square-root of bit flip), X (bit flip), CNOT, and T (general rotation)) guarantees the existence of a universal QCOMP, like a UTM, but may likewise need exponential resources.

11.6.2 Quantum simulation

Feynman (1982) was among the first to point out that quantum systems need to be described by quantum systems. Electronic structure calculation with full account of many-body interactions is $\mathcal{QMA}$-hard (Aaronson, 2009; Schuch and Verstraete, 2009; Whitfield et al., 2013). Therefore, taking an example from biochemistry, to efficiently compute the properties of a catalysing enzyme or the workings of a ribosome will require a quantum simulator to achieve the precision needed in reasonable time.

A QCOMP emulator/simulator is basically an analogue machine: an engineered qubit-array where the interactions between qubits (memory cells) are implemented by substrate-defined or externally induced local and global couplings. The static or quasi-static (adiabatic) interactions can be tuned to implement specific Hamiltonians describing physical models or systems (or even something unphysical), and time-dependent driving will implement dynamic response. All the interactions provide together an effective time-dependent Hamiltonian and a corresponding time-evolution operator $U = exp[-iH_{eff}(t)\ t\ ]$. The induced time-evolution will be characteristic for the system and can be analysed (measured) to provide deterministic or statistical answers to various questions. Note that there is no fundamental difference between digital and analogue QCOMP: if we drive the qubits by, e.g., fast external microwave pulse trains, we can design the time-evolution operator U to generate the specific elementary universal 1q and 2q gates of the quantum circuit model.

Quantum simulation of physical systems (Brown et al., 2010; Georgescu et al., 2014; Wendin, 2017) is now at the focus of intense engineering and experimental efforts (Barends et al., 2015; Barends et al., 2016; Barreiro et al., 2011; Blatt and Roos, 2012; Boixo et al., 2016b; Cirac and Zoller, 2010; Lanyon et al., 2010; O'Malley et al., 2016; Peruzzo et al., 2014; Salathe et al., 2015; Wang et al., 2015) and software development (Bauer et al., 2016; Bravyi and Gosset, 2016; Häner et al., 2016; Häner et al., 2018; Reiher et al., 2017; Valiron et al., 2015; Wecker and Svore, 2014). Materials science and chemistry will present testing grounds for the performance of quantum simulation and computing in the coming years.

11.6.3 Adiabatic quantum optimisation (AQO)

AQO is the analogue version of quantum computing and simulation. It starts from the ground state of a simple known Hamiltonian and slowly (adiabatically) changes the substrate parameters into describing a target Hamiltonian, manoeuvring through the energy landscape, all the time staying in the ground state. The final state and the global minimum then present the solution to the target problem (Farhi et al., 2014; Farhi and Harrow, 2014). AQO is potentially an efficient approach to quantum simulation but has so far been limited to theoretical investigations, e.g., with applications to quantum phase transitions and speed limits for computing. Possibly there is a Quantum No-Free-Lunch theorem stating that digital quantum gate circuits need quantum error correction and AQO needs to manoeuvre adiabatically through a complicated energy landscape, and in the end the computational power is the same.

11.6.4 Quantum annealing (QA)

QA is a version of quantum optimisation where the target Hamiltonian (often a transverse Ising Hamiltonian) is approached while simultaneously lowering the temperature. This is the scheme upon which D-Wave Systems have developed their QA processors, the most recent one built on a chip with a 2000 qubit array and a special cross-bar structure. Despite indications of quantum entanglement and tunnelling within qubit clusters (Boixo et al., 2016a; Denchev et al., 2016), there is no evidence for quantum speed-up (Rønnow et al., 2014; Zintchenko et al., 2015) – so far optimised classical algorithms running on modest classical computers can simulate the quantum annealer.

11.6.5 Quantum machine learning (QML)

QML is an emerging field, introducing adaptive methods from machine language (ML) classical optimisation and neural networks to quantum networks (Aaronson, 2015b; Biamonte et al., 2016; Schuld et al., 2015; Wiebe et al., 2014; Wiebe et al., 2015; Wittek, 2016). One aspect is using ML for optimising classical control of quantum systems. Another, revolutionary, aspect is to apply ML methods to quantum networks for quantum enhanced learning algorithms. The field is rapidly evolving, and we refer to a recent review (Biamonte et al., 2016) for an overview of progress and for references.

11.7 Computational power of classical physical systems and unconventional paradigms

As already mentioned in the Introduction, and discussed to some extent, there is a veritable zoo of UCOMP paradigms (European Commission, 2009). Here we claim that the only decisive borderline is the one that separates classical problems (Newtonian Physics and Relativity) from problems governed by Quantum Physics, which includes some combinatorial problems profiting from the Quantum Fourier Transform (QFT). Quantum information processing and class $\mathcal{BQP}$ is discussed in Sect. 11.6. In this section we focus on a few classical problems of great current interest, representative for the polynomial class $\mathcal{P}$.

There are further issues of measuring the complexity of problems running on UCOMP devices. The actions of the device might not map well to the parameters of time, space, and problem size needed for classical complexity analysis. And the computation may use resources not considered in classical complexity analysis, for example, the time needed to read out a result.

11.7.1 DNA computing

Computing with DNA or RNA strands was first investigated theoretically. Bennett (1982) imagines a DTM built from RNA reactions:

> The tape might be a linear informational macromolecule analogous to RNA, with an additional chemical group attached at one site to encode the head state [. . .] and location. Several hypothetical enzymes (one for each of the Turing Machine's transition rules) would catalyse reactions of the macromolecule with small molecules in the surrounding solution, transforming the macromolecule into its logical successor.

Shapiro (2012) proposes a more detailed design for a general purpose polymer-based DTM. Qian et al. (2010) describe a DNA-based design for a stack machine. These designs demonstrate that general purpose polymer-based computing is possible, at least in principle. None of these designs challenge the ECT: they are all for DTMs or equivalent power machines.

Adleman (1994) was the first to implement a form of DNA computing in the wetlab, with his seminal paper describing the solution to a 7-node instance of the Hamiltonian path problem. This is an $\mathcal{NP}$-complete decision problem on a graph: is there a path through a graph that visits each node exactly once? Adleman's approach encodes the graph nodes and edges using small single strands of DNA, designed so that the edges can stick to the corresponding vertices by complementary matching. Sufficient strands are put into a well mixed system, and allowed to stick together. A series of chemical processes are used to extract the resulting DNA, and to search for a piece that has encoded a solution to the problem. The time taken by these processes is linear in the number of nodes, but the number of strands needed to ensure the relevant ones meet and stick with high enough probability grows exponentially with the number of nodes (Adleman, 1994). Essentially, this set-up needs enough DNA to construct all possible paths, to ensure that the desired solution path is constructed. So this algorithm solves the problem in polynomial time, by using massive parallelism, at the cost of exponential DNA resources (and hence exponential space). Hartmanis (1995) calculates that this form of computation of the Hamiltonian path on a graph with 200 nodes would need a mass of DNA greater than that of the earth.

Lipton (1995) critiques Adleman's algorithm, because it is "brute force" in trying all possible paths. He describes a (theoretical) DNA algorithm to solve the $\mathcal{NP}$-complete SAT problem (Sect. 11.2.2). For a problem of n Boolean variables and m clauses, the algorithm requires a number of chemical processing steps linear in m. However, it also requires enough DNA to encode "all possible n-bit numbers" (that is, all possible assignments of the n Boolean variables), so it also requires exponential DNA resources.

So these special-purpose forms of DNA computing, focussed on $\mathcal{NP}$-complete problems, trade off exponential time for exponential space (massively parallel use of DNA resources). Additionally, it is by no means clear that the chemical processing and other engineering facilities would remain polynomial in time once the exponential physical size of the reactants kicks into effect.

Other authors consider using the informational and constructive properties of DNA for a wide range of computational purposes. For example, implementations of tiling models often use DNA to construct the tiles and program their connections.

The Wang tile model (Wang, 1961) has been used to show theorem proving and computational capabilities within, e.g., DNA computing. It was introduced by Wang, who posed several conjectures and problems related to the question whether a given finite set of Wang tiles can tile the plane. Wang's student Robert Berger showed how to emulate any DTM by a finite set of Wang tiles (Berger, 1966). Using this, he proved that the undecidability of the Halting Problem implies the undecidability of Wang's tiling problem.

Later applications demonstrate the (computational) power of tiles; see, e.g., Yang (2013, Chap. 6). In particular, $\mathcal{NP}$-complete problems like k-SAT have been solved in linear time in the size of the input using a finite number of different tiles (Brun, 2008). The hidden complexity lies in the exponentially many parallel tile assemblies (the computation is nondeterministic and each parallel assembly executes in time linear in the input size).

As for the latter example, in each of these models the complexity properties need to be carefully established. Properties established in one implementation approach may not carry over to a different implementation. For example, Seelig and Soloveichik (2009) consider molecular logic circuits with many components arranged in multiple layers built using DNA strand displacement; they show that the time-complexity does not necessarily scale linearly with the circuit depth, but rather can be quadratic, and that catalysis can alter the asymptotic time-complexity.

11.7.2 Networks of evolutionary processors

An "Accepting Hybrid Network of Evolutionary Processors" (AHNEP) is a theoretical device for exploring language-accepting processes. Castellanos et al. (2001) describe the design. It comprises a fully connected graph. Each graph node contains: (i) a simple evolutionary processor that can perform certain point mutations (insertion, deletion, substitution) on data, expressed as rewrite rules; (ii) data in the form of a multiset of strings, which are processed in parallel such that all possible mutations that can take place do so. In particular, if a specified substitution may act on different occurrences of a symbol in a string, each occurrence is substituted in a different copy of the string. For this to be possible, there is an arbitrarily large number of copies of each string in the multiset. The data moves through the network; it must pass a filtering process that depends on conditions of both the sender and receiver.

Castellanos et al. (2001) demonstrate that such networks of linear size (number of nodes) can solve $\mathcal{NP}$-complete problems in linear time. Much subsequent work has gone into variants (Margenstern et al., 2005), and determining bounds on the size of such networks. For example, Manea and Mitrana (2007) find a constant size network of 24 nodes that can solve $\mathcal{NP}$ problems in polynomial time; Loos et al. (2009) reduce that bound to 16 nodes. Alhazov et al. (2014) prove that a 5 node AHNEP is computationally complete.

Note that some of the papers referenced demonstrate that AHNEPs can solve $\mathcal{NP}$-complete problems in polynomial time. They manage to do so in the same way the DNA computers of the previous section do: by exploiting exponential space resources. The set-up exploits use of an exponentially large data set at each node, by requiring an arbitrarily large number of each string be present, so that all possible substitutions can occur in parallel.

11.7.3 Evolution in materio

Evolution *in materio* (EiM) is a term coined by Miller and Downing (2002) to refer to material systems that can be used for computation by manipulating the state of the material through external stimuli, e.g., voltages, currents, optical signals and the like, and using some fixed input and output channels to the material for defining the wanted functionality. In EiM, the material is treated as a black box, and computer-controlled evolution (by applying genetic algorithms or other optimisation techniques, using a digital computer) is used to change the external stimuli (the configuration signals) in such a way that the black box converges to the target functionality, i.e., the material system produces the correct output combinations, representing solutions to the problem when certain input combinations, representing problem instances of the problem, are applied. Experimental results show that this approach has successfully been applied to different types of problems, with different types of materials (Broersma et al., 2016), but mainly for either small instances of problems or for rather simple functionalities like Boolean logic gates. Interestingly, using EiM, reconfigurable logic has been evolved in a stable and reproducible way on disordered nanoparticle networks of very small size, comparable to the size that would be required by arrangements of current transistors to show the same functionality (Bose et al., 2015).

It is impossible and it would not be fair to compare the complexity of the solution concept of EiM to that of classical computation. First of all, apart from the (digital) genetic algorithms or other optimisation techniques that are used to manipulate the system, there is no algorithmic procedure involved in the actual computation. The material is not executing a program to solve any particular problem instance; instead, a set of problem instances and their target outputs are used in the evolutionary process of configuring the material. In that sense, the material is more or less forced to produce the correct (or an approximate solution that can be translated into a correct) solution for that set of problem instances. If this is not the whole set of possible instances, there is no guarantee that the material outputs the correct solution for any of the other problem instances. In fact, since fitness functions are used to judge the quality of the configurations according to the input-output combinations they produce, even the correctness of the output for individual instances that are used during the evolutionary process is questionable, unless they are checked one by one at the end of this process. In a way, for problems with an unbounded number of possible instances, the EiM approach can be regarded as a heuristic without any performance guarantees for the general problem. So, it is not an alternative to exact solution concepts from classical computation, and hence it cannot claim any particular relevance for (exactly) solving $\mathcal{NP}$-hard or $\mathcal{NP}$-complete problems, let alone undecidable problems.

Secondly, in EiM the time it takes for the evolutionary process to converge to a satisfactory configuration of the material for solving a particular problem is the crucial measure in terms of time complexity. After that, the material system does, in principle, produce solutions to instances almost instantaneously. In a sense, this evolutionary process can be regarded as a kind of preprocessing, but different from the preprocessing that is used in classical computation to decrease the size of the instances, an important concept in the domain of FPT. Clearly, there are issues with scalability involved in EiM. It is likely that a limited amount of material, together with a limited amount of input and output channels, and a limited amount of configuration signals, has a bounded capability of solving instances of a problem with an unbounded number of possible instances. It seems difficult to take all of these aspects into account in order to define a good measure for the capability of EiM to tackle hard problems. Such problems are perhaps not the best candidates for the EIM approach. Instead, it might be better to focus future research on computational tasks that are difficult to accomplish with classical computational devices; not difficult in the sense of computational complexity but in the sense of developing and implementing the necessary computer programs to perform the tasks. One might think of classification tasks like speech, face and pattern recognition.

11.7.4 Optical computing

It is possible to use optical systems to compute with photons rather than electrons. Although this is a somewhat unconventional computational substrate, the computation performed is purely classical.

Some authors suggest more unconventional applications of optical components. Woods and Naughton (2005) and Woods and Naughton (2009) discuss a form of spatial optical computing that encodes data as images, and computes by transforming the images through optical operations. These include both analogue and digital encodings of data.

Reif et al. (1994) describe a particular idealised ray tracing problem cast as a decision problem, and show that it is Turing-uncomputable. They also note that the idealisations do not hold in the physical world:

> Theoretically, these optical systems can be viewed as general optical computing machines, if our constructions could be carried out with infinite precision, or perfect accuracy. However, these systems are not practical, since the above assumptions do not hold in the physical world. Specifically, since the wavelength of light is finite, the wave property of light, namely diffraction, makes the theory of geometrical optics fail at the wavelength level of distances.

Blakey (2014) has furthermore formalised the intuition that the claimed hypercomputational power of even such idealised computers in fact vanishes once precision is properly considered.

Wu et al. (2014) construct an optical network than can act as an oracle for the Hamiltonian path decision problem. Their encoding approach addresses the usual precision issue by having exponentially large delays; hence, as they say, it does not reduce the complexity of the problem, still requiring exponential time. They do however claim that it can provide a substantial speed-up factor over traditional algorithms, and demonstrate this on a small example network. However, since the approach explores all possible paths in parallel, this implies an exponential power requirement, too.

11.7.5 MEM-computing

MEM-computing has been introduced by Traversa and Di Ventra (2015), Traversa and Di Ventra (2017), and Traversa et al. (2015) as a novel non-Turing model of computation that uses interacting computational memory cells – memprocessors – to store and process information in parallel on the same physical platform, using the topology of the interaction network to form the specific computation.

Traversa and Di Ventra (2015) introduce the Universal Memcomputing Machine (UMM), and make a strong claim:

> We also demonstrate that a UMM has the same computational power as a non-deterministic Turing machine, namely it can solve $\mathcal{NP}$-complete problems in polynomial time. However, by virtue of its information overhead, a UMM needs only an amount of memory cells (memprocessors) that grows polynomially with the problem size. ... Even though these results do not prove the statement $\mathcal{NP} = \mathcal{P}$ within the Turing paradigm, the practical realization of these UMMs would represent a paradigm shift from present von Neumann architectures bringing us closer to brain-like neural computation.

The UMM is essentially an analogue device. Traversa and Di Ventra (2015) state:

a UMM can operate, in principle, on an infinite number of continuous states, even if the number of memprocessors is finite. The reason being that each memprocessor is essentially an analog device with a continuous set of state values

then in a footnote acknowledge:

Of course, the actual implementation of a UMM will limit this continuous range to a discrete set of states whose density depends on the experimental resolution of the writing and reading operations.

Traversa et al. (2015) present an experimental demonstration with 6 memprocessors solving a small instance of the NP-complete version of the subset sum problem in only one step. The number of memprocessors in the given design scales linearly with the size of the problem. The authors state:

the particular machine presented here is eventually limited by noise—and will thus require error-correcting codes to scale to an arbitrary number of memprocessors

Again, the issue is an analogue encoding, here requiring a Fourier transform of an exponential number of frequencies, and accompanying exponential requirements on precision, and possibly power. As we have emphasised earlier (Sect. 11.5.1), complexity is an asymptotic property, and small problem instances do not provide evidence of asymptotic behaviour. See also Saunders (2016) for a further critique of this issue. Traversa et al. (2015) state that this demonstration experiment "represents the first proof of concept of a machine capable of working with the collective state of interacting memory cells", exploring the exponentially large solution space in parallel using waves of different frequencies. Traversa et al. (2015) suggest that this is similar to what quantum computing does when solving difficult problems such as factorisation. However, although quantum computing is a powerful and physical model, there is no evidence that it can efficiently solve $\mathcal{NP}$-hard problems. The power of quantum computing is quantum superposition and entanglement in Hilbert space, not merely classical wave computing using collective coherent classical states. The oracles needed for a quantum computer to solve problems in $\mathcal{QMA}$ most likely do not exist in the physical world. Aaronson (2005) examines and refutes many claims to solving $\mathcal{NP}$-hard problems, demonstrating the different kinds of smuggling that are used, sweeping the eventual need for exponential resources under (very large) carpets. This is also underlined in Aaronson (2015a), a blog post providing an extended critique of the claims in Traversa et al. (2015).

In order to address these scaling limitations of the analogue UMM, Traversa and Di Ventra (2017) present their *Digital* Memcomputing Machine (DMM). They claim that DMMs also have the computational power of nondeterministic Turing machines, able to solve $\mathcal{NP}$-complete problems in polynomial time with resources that scale polynomially with the input size. They define a dynamical systems model of the DMM, and prove several properties of the model. They provide the results of several numerical simulations of a DMM circuit solving small instances of an $\mathcal{NP}$-complete problem, and include a discussion of how their results suggest, though do not prove, that $\mathcal{P} = \mathcal{NP}$. The computational power of the DMM is claimed to arise from its "intrinsic parallelism" of operation. This intrinsic parallelism is a consequence of the DMM components communicating with each other in an analogue manner during a computational state transition step. The DMM is digital insofar as it has a digital state at the beginning and end of a computational step. However, its operation has the same fundamentally analogue nature as the UMM during the intrinsically parallel execution of a computational step, and so all the issues of precision and noise will still need to be addressed before any super-Turing properties can be established.

11.7.6 Brain computing

The brain is generally considered to be a natural physical adaptive information processor subject to physical law; see, e.g., Bassett and Gazzaniga (2011), Chesi and Moro (2014), Juba (2016), and Schaul et al. (2011). As such it must essentially be a classical analogue "machine", and then the ECT states that it can, in principle, be simulated efficiently by a digital classical computer.

This "limitation" of the brain is not always accepted, however, especially by philosophers. Leaving aside far-fetched quantum-inspired models, brain-inspired models sometimes assume that the brain is able to efficiently solve $\mathcal{NP}$-hard problems, see e.g. Traversa et al. (2015), and therefore can serve as a model for super-Turing computing beyond classical digital DTMs. The underlying idea is likely that (human) intelligence and consciousness are so dependent on processing power that classical digital computers cannot efficiently model an intelligent self-conscious brain – such problems must necessarily be $\mathcal{NP}$-hard.

The view of this chapter's authors is that our brain (actually any animal brain) is a powerful but classical information processor, and that its workings remain to be found out. The fact that we have no real understanding of the difference between a conscious and unconscious brain is most likely not to be linked to the lack of processing capacity.

One side of the problem-solving capacity of the brain is demonstrated in game playing, as illustrated by the performance of the AlphaGo adaptive program of Google DeepMind (Silver et al., 2016), run on a digital computer and winning over both the European and the world champions (Metz, 2016), beating the human world champion 4–1 in a series of Go games. AlphaGo is adaptive, based on deep neural networks (machine learning) and tree search, probably representing a significant step toward powerful artificial intelligence (AI). Since the human champion no doubt can be called intelligent, there is some foundation for ascribing some intelligence to the AlphaGo adaptive program. The problem of characterising artificial intelligence can be investigated via game playing (Schaul et al., 2011).

Games present $\mathcal{NP}$-hard problems when scaled up (Viglietta, 2012). Aloupis et al. (2015) have proved the $\mathcal{NP}$-hardness of five of Nintendo's largest video game franchises. In addition, they prove $\mathcal{PSPACE}$-completeness of the Donkey Kong Country games and several Legend of Zelda games.

For AlphaGo not only to show some intelligence, but also be aware of it, i.e., aware of itself, is of course quite a different thing. This does not necessarily mean that consciousness presents a dramatically harder computational task. But it then needs mechanisms for self-observation and at least short-term memory for storing those observations. When AlphaGo then starts describing why it is making the various moves, thinking about them (!), and even recognising its errors, then one may perhaps grant it some consciousness, and perhaps even intuition.

The argument that the brain violates the ECT appears to rest on the fact that brains can solve certain problems that are uncomputable or $\mathcal{NP}$-hard. However, this argument confuses the *class* of problems that have these properties with the individual instances that we can solve. So, for example, despite our being able to prove whether *certain* programs terminate, there is no evidence that we can prove whether *any* program terminates.

We hold that the brain is a natural, physical, basically analogue information processor that *cannot* solve difficult instances of $\mathcal{NP}$-hard problems. Therefore intelligence, consciousness, intuition, etc. must be the result of natural computation processes that, in principle, can be *efficiently* simulated by a classical digital computer, given some models for the relevant brain circuitry. The question of polynomial overhead is a different issue. Perhaps it will be advantageous, or even necessary, to emulate some brain circuits in hardware in order to get reasonable response times of artificial brain models. Such energy-efficient neuromorphic hardware is already emerging (Esser et al., 2016; Merolla et al., 2014) and may soon match human recognition capabilities (Maass, 2016).

11.7.7 Conclusion

These discussed forms of unconventional physically realisable computing are unable to solve $\mathcal{NP}$-hard problems in polynomial time, unless given access to an uncomputable oracle or some "smuggled" exponential resource, such as precision, material, or power. So, hard instances of $\mathcal{NP}$ problems cannot be solved efficiently. Various approaches to solving $\mathcal{NP}$-hard problems result in at best polynomial speed-up. The quantum class $\mathcal{BQP}$ does indicate that quantum computers can offer efficiency improvements for some problems outside the $\mathcal{NP}$-complete class (always assuming $\mathcal{P} \neq \mathcal{NP}$).

11.8 Overall conclusions and perspectives

We have provided an overview of certain claims of hypercomputation and super-Turing computation in a range of unconventional computing devices. Our overview covers many specific proposals, but is not comprehensive. Nevertheless, a common theme emerges: all the devices seem to rely on one or another *unphysical* property to work: infinite times or speeds, or infinite precision, or uncomputable advice, or some unconsidered exponential physical resource.

There is value in examining a wide range of unconventional theoretical models of computation, even if those models turn out to be unphysical. After all, even the theoretical model that is the DTM is unphysical: its unbounded memory tape, necessary for its theoretical power, cannot be implemented in our finite bounded universe. Our physical digital computers are all finite state machines. Claims about *physical* implementability of models more powerful than the DTM, when the DTM itself is unphysical, need to be scrutinised carefully, and not in the realm of mathematics (their theoretical power), but rather in that of physics (their implementable power).

One issue that UCOMP helps foreground is this existence of *physical* constraints on *computational* power (Aaronson, 2005; Aaronson, 2009; Aaronson, 2013b; Aaronson, 2015a; Denef and Douglas, 2007; Potgieter, 2006). That there might be such physical limits to computational power may be difficult to accept, judging from a wealth of publications discussing and describing how to efficiently solve $\mathcal{NP}$-hard problems with physical computers. However, as we have argued in this chapter, there is no convincing evidence that classical (non-quantum) computational devices (whether analogue or digital, engineered or evolved) can be built to *efficiently* solve problems outside classical complexity class $\mathcal{P}$.

The Laws of Thermodynamics, which address energy conservation and entropy increase, express bounds on what is physically possible. A consequence of these laws is that perpetual motion machines are physically *impossible*, and any purported design will have a flaw somewhere. These are laws of physics, not provable mathematical theorems, but are well-evidenced. The laws are flexible enough that newly discovered phenomena (such as the convertibility of mass and energy) can be accommodated.

The CTT and ECT appear to play an analogous role in computation. They express bounds on what computation is *physically* possible. A consequence of these bounds, if they are true, is that hypercomputing and super-Turing computing are *physically* impossible. And they are similarly flexible that newly discovered phenomena (such as quantum computing) can be accommodated. This demonstrates a fundamental and deep connection between computation and the laws of physics: computation can be considered as a natural science, constrained by reality, not as an abstract branch of mathematics (Horsman et al., 2017).

The CTT and ECT can be expressed in a, slightly tongue-in-cheek, form echoing the laws of thermodynamics:

1st Law of Computing: You cannot solve uncomputable or $\mathcal{NP}$-hard problems efficiently unless you have a physical infinity or an efficient oracle.
2nd Law of Computing: There are no physical infinities or efficient oracles.
3rd Law of Computing: Nature is physical and does not solve uncomputable or $\mathcal{NP}$-hard problems efficiently.
Corollary: Nature necessarily solves uncomputable or $\mathcal{NP}$-hard problems only approximately.

This raises the question: what can UCOMP do? Given that UCOMP does not solve uncomputable or even $\mathcal{NP}$-hard problems (and this also applies to quantum computing), what is the future for UCOMP? Instead of simply trying to do conventional things faster, UCOMP can focus on novel applications, and novel insights into computation, including:

- Insight into the relationship between physics and computation
- New means for analogue simulation/optimisation
- Huge parallelisation
- Significant polynomial speed-up
- Novel forms of approximate solutions
- Cost-effective solutions
- Solutions beyond the practical capability of digital HPC
- Solutions in novel physical devices, for example, programmed synthetic biological cells

UCOMP offers many things. But it does not offer hypercomputing or super-Turing computing realisable in the physical world.

Acknowledgements

H.B. acknowledges funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 317662 (the FP7 FET NASCENCE project). S.S. acknowledges partial funding by the EU FP7 FET Coordination Activity TRUCE (Training and Research in Unconventional Computation in Europe), project reference number 318235. G.W. acknowledges support by the European Commission through the FP7 SYMONE and Horizon 2020 RECORD-IT projects, and by Chalmers University of Technology.

References

Aaronson, Scott (2005). “NP-complete problems and physical reality”. *ACM SIGACT News* 36(1):30–52.

Aaronson, Scott (2008). “The Limits of Quantum Computers”. *Scientific American* 298(3):62–69.

Aaronson, Scott (2009). “Computational complexity: Why quantum chemistry is hard”. *Nature Physics* 5(10):707–708.

Aaronson, Scott (2013a). *Quantum Computing Since Democritus*. Cambridge University Press.

Aaronson, Scott (2013b). “Why Philosophers Should Care About Computational Complexity”. *Computability: Gödel, Turing, Church, and Beyond*. Ed. by B. J. Copeland, C. J. Posy, and O. Shagrir. MIT Press.

Aaronson, Scott (2015a). *Memrefuting*. www.scottaaronson.com/blog/?p=2212.

Aaronson, Scott (2015b). “Read the fine print”. *Nature Physics* 11:291–293.

Aaronson, Scott (2017). “P =? NP”. www.scottaaronson.com/papers/pnp.pdf.

Adamatzky, Andrew, ed. (2017). *Advances in Unconventional Computing, vol 1*. Springer.

Adleman, Leonard M. (1994). “Molecular computation of solutions to combinatorial problems”. *Science* 266:1021–1024.

Alhazov, A., R. Freund, and Y. Rogozhin (2014). “Five nodes are sufficient for hybrid networks of evolutionary processors to be computationally complete”. *UCNC 2014*. Vol. 8553. LNCS. Springer, pp. 1–13.

Aloupis, G., E. D. Demaine, A. Guo, and G. Viglietta (2015). “Classic Nintendo games are (computationally) hard”. *Theoretical Computer Science* 586:135–160.

Ambaram, Tânia, Edwin Beggs, José Félix Costa, Diogo Poças, and John V. Tucker (2017). “An Analogue-Digital Model of Computation: Turing Machines with Physical Oracles”. *Advances in Unconventional Computing, vol 1*. Ed. by Andrew Adamatzky. Springer, pp. 73–115.

Arkoudas, K. (2008). “Computation, hypercomputation, and physical science”. *Journal of Applied Logic* 6(4):461–475.

Balcázar, José and Montserrat Hermo (1998). “The Structure of Logarithmic Advice Complexity Classes”. *Theoretical Computer Science* 207:217–244.

Barends, R., L. Lamata, J. Kelly, L. García-Álvarez, A. G. Fowler, A. Megrant, E. Jeffrey, T. C. White, D. Sank, J. Y. Mutus, B. Campbell, Yu Chen, Z. Chen, B. Chiaro, A. Dunsworth, I.-C. Hoi, C. Neill, P. J. J. O’Malley, C. Quintana, P. Roushan, A. Vainsencher, J. Wenner, E. Solano, and John M. Martinis (2015). “Digital quantum simulation of fermionic models with a superconducting circuit”. *Nature Commun.* 6:7654.

Barends, R., A. Shabani, L. Lamata, J. Kelly, A. Mezzacapo, U. Las Heras, R. Babbush, A. G. Fowler, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A.

Dunsworth, E. Jeffrey, E. Lucero, A. Megrant, J. Y. Mutus, M. Neeley, C. Neill, P. J. J. O'Malley, C. Quintana, P. Roushan, D. Sank, A. Vainsencher, J. Wenner, T. C. White, E. Solano, H. Neven, and John M. Martinis (2016). "Digitized adiabatic quantum computing with a superconducting circuit". *Nature* 534:222–226.

Barreiro, J. T., M. Müller, P. Schindler, D. Nigg, T. Monz, M. Chwalla, M. Hennrich, C. F. Roos, P. Zoller, and R. Blatt (2011). "An open-system quantum simulator with trapped ions". *Nature* 470:486–491.

Bassett, D. S. and M. S. Gazzaniga (2011). "Understanding complexity in the human brain". *Trends in Cognitive Sciences* 15(5):200–209.

Bauer, Bela, Dave Wecker, Andrew J. Millis, Matthew B. Hastings, and Matthias Troyer (2016). "Hybrid Quantum-Classical Approach to Correlated Materials". *Phys. Rev. X* 6(3):031045.

Beggs, Edwin, José Félix Costa, Diogo Poças, and John V. Tucker (2014). "An Analogue-Digital Church-Turing Thesis". *International Journal of Foundations of Computer Science* 25(4):373–389.

Bennett, Charles H. (1982). "The thermodynamics of computation—a review". *International Journal of Theoretical Physics* 21(12):905–940.

Berger, Robert (1966). "The undecidability of the domino problem". *Memoirs of the American Mathematical Society* 66:1–72.

Biamonte, Jacob, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd (2016). *Quantum Machine Learning*. eprint: arXiv:1611.09347[quant-ph].

Blakey, Ed (2014). "Ray tracing – computing the incomputable?" *Proc DCM 2012*. Ed. by B. Löwe and G. Winskel. Vol. 143. EPTCS, pp. 32–40.

Blakey, Ed (2017). "Unconventional Computers and Unconventional Complexity Measures". *Advances in Unconventional Computing, vol 1*. Ed. by Andrew Adamatzky. Springer, pp. 165–182.

Blatt, R. and C. F. Roos (2012). "Quantum simulations with trapped ions". *Nature Physics* 8:277–284.

Blum, L. (2004). "Computing over the reals: Where Turing Meets Newton". *Notices of the AMS* 51(9):1024–1034.

Boixo, S., V. N. Smelyanskiy, A. Shabani, S. V. Isakov, M. Dykman, V. S. Denchev, M. Amin, A. Smirnov, M. Mohseni, and H. Neven (2016a). "Computational Role of Multiqubit Tunneling in a Quantum Annealer". *Nature Commun.* 7:10327.

Boixo, Sergio, Sergei V. Isakov, Vadim N. Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, John M. Martinis, and Hartmut Neven (2016b). *Characterizing Quantum Supremacy in Near-Term Devices*. eprint: arXiv:1608.00263[quant-ph].

Bose, S. K., C. P. Lawrence, Z. Liu, K. S. Makarenko, R. M. J. van Damme, H. J. Broersma, and W. G. van der Wiel (2015). "Evolution of a designless nanoparticle network into reconfigurable Boolean logic". *Nature Nanotechnology* 10:1048–1052.

Bravyi, S. and D. Gosset (2016). "Improved Classical Simulation of Quantum Circuits Dominated by Clifford Gates". *Phys. Rev. Lett.* 116:250501.

Broersma, H . J., J. F. Miller, and S. Nichele (2016). "Computational Matter: Evolving Computational Functions in Nanoscale Materials". *Advances in Unconventional Computing Volume 2: Prototypes, Models and Algorithms.* Ed. by A. Adamatzky, pp. 397–428.

Brown, K. L., W. J. Munro, and V. M. Kendon (2010). "Using Quantum Computers for Quantum Simulation". *Entropy* 12:2268–2307.

Brun, Yuriy (2008). "Solving satisfiability in the tile assembly model with a constant-size tileset". *Journal of Algorithms* 63(4):151–166.

Bryngelson, J. D., J. Nelson Onuchic, N. D. Socci, and P. G. Wolynes (1995). "Funnels, pathways, and the energy landscape of protein folding: A synthesis". *Proteins: Structure, Function and Bioinformatics* 21(3):167–195.

Cabessa, J. and H. T. Siegelmann (2011). "Evolving recurrent neural networks are super-Turing". *Proc. IJCNN 2011.* IEEE, pp. 3200–3206.

Castellanos, Juan, Carlos Martín-Vide, Victor Mitrana, and José M. Sempere (2001). "Solving NP-complete problems with networks of evolutionary processors". *IWANN 2001.* Vol. 2084. LNCS. Springer, pp. 621–628.

Chaitin, Gregory J. (1975). "A Theory of Program Size Formally Identical to Information Theory". *Journal of the ACM* 22(3):329–340.

Chaitin, Gregory J. (2012). "How Much Information Can There Be in a Real Number?" *Computation, Physics and Beyond.* Springer, pp. 247–251.

Cheeseman, Peter, Bob Kanefsky, and William M Taylor (1991). "Where the Really Hard Problems Are". *Proc. IJCAI 1991.* Morgan Kaufmann, pp. 331–337.

Chesi, C. and A. Moro (2014). "Computational complexity in the brain". *Measuring Grammatical Complexity.* Ed. by F. J. Newmeyer and L. B. Preston. Oxford University Press, pp. 264–280.

Cirac, J. I. and P. Zoller (2010). "Goals and opportunities in quantum simulation". *Nature Phys.* 8:264–266.

Complexity Zoo website (n.d.). complexityzoo.uwaterloo.ca/Complexity_Zoo.

Cook, Stephen (1971). "The complexity of theorem proving procedures". *Proceedings of the Third Annual ACM Symposium on Theory of Computing* :151–158.

Copeland, B. J. (2004). "Hypercomputation: philosophical issues". *Theoretical Computer Science* 317(1–3):251–267.

Copeland, B. J. (2015). "The Church-Turing Thesis". *The Stanford Encyclopedia of Philosophy.* Ed. by Edward N. Zalta. Summer 2015.

Copeland, B. J. and O. Shagrir (2011). "Do accelerating Turing machines compute the uncomputable?" *Minds and Machines* 21(2):221–239.

Crescenzi, P. and V. Kann (2005). *A compendium of NP optimization problems.* www.nada.kth.se/~viggo/problemlist/compendium.html.

Denchev, V. S., S. Boixo, S. V. Isakov, N. Ding, R. Babbush, J. Martinis V. Smelyanskiy and, and H. Neven (2016). "What is the Computational Value of Finite Range Tunneling?" *Phys. Rev. X* 6:031015.

Denef, F. and M. R. Douglas (2007). "Computational complexity of the landscape: Part I". *Annals of Physics* 322(5):1096–1142.

Dershowitz, N. and E. Falkovich (2012). "A Formalization and Proof of the Extended Church-Turing Thesis". *EPTCS* 88:72–78.

Deutsch, David (1985). "Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer". *Proceedings of the Royal Society A* 400(1818):97–117.

Douglas, Keith (2003). "Super-Turing Computation: A Case Study Analysis". www.philosopher-animal.com/papers/take6c.pdf. MA thesis. Carnegie Mellon University.

Douglas, Keith (2013). "Learning to Hypercompute? An Analysis of Siegelmann Networks". *Computing Nature: Turing Centenary Perspective.* Ed. by G. Dogic-Crnkovic and R. Giovagnoli. Springer, pp. 201–211.

Downey, Rod G. and Michael R. Fellows (1999). *Parameterized Complexity.* Springer.

Edmonds, Jack (1965). "Paths, trees and flowers". *Canadian J. of Math.* 17 :449–467.

Esser, S. K., P. A. Merolla, A. S. Cassidy J. V. Arthur and, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, C. di Nolfo, P. Datta, A. Amir, B. Taba, M. D. Flickner, and D. S. Modha (2016). "Convolutional networks for fast, energy-efficient neuromorphic computing". *PNAS* 41(113):668–673.

Etesi, Gábor and István Németi (2002). "Non-Turing Computations Via Malament–Hogarth Space-Times". *International Journal of Theoretical Physics* 41(2):341–370.

European Commission (2009). "Unconventional Formalisms for Computation". *Expert Consultation Workshop*.

Farhi, Edward, Jeffrey Goldstone, and Sam Gutmann (2014). *A Quantum Approximate Optimization Algorithm.* eprint: arXiv:1411.4028[quant-ph].

Farhi, Edward and Aram W. Harrow (2014). *Quantum Supremacy through the Quantum Approximate Optimization Algorithm.* eprint: arXiv:1602.07674[quant-ph].

Feynman, R. P. (1982). "Simulating Physics with Computers". *Int. J. Theor. Phys.* 21(6/7):467–488.

Floyd, Robert W. (1967). "Nondeterministic Algorithms". *Journal of the ACM* 14(4):636–644.

Garey, M. R. and D. S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman.

Georgescu, I. M., S. Ashhab, and Franco Nori (2014). "Quantum simulation". *Rev. Mod. Phys.* 86:153–185.

Grover, L. K. (1996). "A fast quantum mechanical algorithm for database search". *Proc. 28th Annual ACM Symposium on the Theory of Computing.* ACM Press, pp. 212–219.

Häner, Thomas, Damian S. Steiger, Mikhail Smelyanskiy, and Matthias Troyer (2016). "High Performance Emulation of Quantum Circuits". *Proc. SC 2016.* eprint: arXiv:1604.06460[quant-ph].

Häner, Thomas, Damian S Steiger, Krysta Svore, and Matthias Troyer (2018). "A software methodology for compiling quantum programs". *Quantum Science and Technology* 3(2):020501.

Hartmanis, Juris (1995). "The structural complexity column: On the Weight of Computations". *Bulletin of the European Association for Theoretical Computer Science. EATCS* 55:136–138.

Hogarth, Mark L. (1992). "Does General Relativity Allow an Observer to View an Eternity in a Finite Time?" *Foundations of Physics Letters* 5 :173–181.

Horodecki, R., P. Horodecki, M. Horodecki, and K. Horodecki (2009). "Quantum entanglement". *Rev. Mod. Phys* 81(2):865–942.

Horsman, C., Susan Stepney, Rob C. Wagner, and Viv Kendon (2014). "When does a physical system compute?" *Proceedings of the Royal Society A* 470(2169):20140182.

Horsman, Dominic, Susan Stepney, and Viv Kendon (2017). "The Natural Science of Computation". *Communications of ACM* 60(8):31–34.

Jozsa, R. and N. Linden (2003). "On the role of entanglement in quantum-computational speed-up". *Proc. R. Soc. Lond. A* 459:2011–2032.

Juba, B. (2016). "Computational complexity and the Function-Structure-Environment Loop of the Brain". *Closed-Loop Neuroscience.* Ed. by A. El Hady. Academic Press, pp. 131–144.

Karp, Richard M. (1972). "Reducibility Among Combinatorial Problems". *Complexity of Computer Computations.* Ed. by R. E. Miller and J. W. Thatcher. Plenum, pp. 85–103.

Konkoli, Z. and G. Wendin (2014). "On information processing with networks of nano-scale switching elements". *Int. Journal of Unconventional Computing* 10(5–6):405–428.

Lanyon, B. P., J. D. Whitfield, G. G. Gillett, M. E. Goggin, M. P. Almeida, I. Kassal, J. D. Biamonte, M. Mohseni, B. J. Powell, M. Barbieri, A. Aspuru-Guzik, and A. G. White (2010). "Towards quantum chemistry on a quantum computer". *Nature Chemistry* 2:106–111.

Lawler, Eugene L., Jan Karel Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys (1985). *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* Wiley.

Levin, Leonid (1973). "Universal search problems". *Problems of Information Transmission* 9(3):(in Russian), 115–116.

Li, Ming and Paul Vitányi (1997). *An Introduction to Kolmogorov Complexity and Its Applications.* 2nd edn. Springer.

Lipton, Richard J. (1995). "DNA solution of hard computational problems". *Science* 268(5210):542–545.

Lloyd, Seth (2000). "Ultimate physical limits to computation". *Nature* 406(6799):1047–1054.

Loos, Remco, Florin Manea, and Victor Mitrana (2009). "Small universal accepting networks of evolutionary processors with filtered connections". *11th International Workshop on Descriptional Complexity of Formal Systems*. Ed. by J. Dassow, G. Pighizzini, and B. Truthe. EPTCS 3, pp. 173–182. eprint: arXiv:0907.5130[cs.FL].

Maass, Wolfgang (2016). "Energy-efficient neural network chips approach human recognition capabilities". *PNAS* 113(41):11387–11389.

Manea, F. and V. Mitrana (2007). "All NP-problems can be solved in polynomial time by accepting hybrid networks of evolutionary processors of constant size". *Information Processing Letters* 103(3):112–118.

Margenstern, Maurice, Victor Mitrana, and Mario J Pérez-Jiménez (2005). "Accepting Hybrid Networks of Evolutionary Processors". *Proc. DNA 2004*. Vol. 3384. LNCS. Springer, pp. 235–246.

Merolla, P. A., J. V. Arthur, A. S. Cassidy R. Alvarez-Icaza and, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha (2014). "A million spiking-neuron integrated circuit with a scalable communication network and interface". *Science* (345):668–673.

Metz, C. (2016). *Google's AI Wins Fifth And Final Game Against Go Genius Lee Sedol*. www.wired.com/2016/03/googles-ai-wins-fifth-final-game-go-genius-lee-sedol/.

Miller, Julian F. and Keith Downing (2002). "Evolution in materio: Looking beyond the silicon box". *Proc. NASA/DoD Evolvable Hardware Workshop*, pp. 167–176.

Montanaro, A. (2016). "Quantum algorithms: an overview". *npj Quantum Information* 2:15023.

Moore, Gordon (1965). "Cramming More Components onto Integrated Circuits". *Electronics Magazine* 38(8):114–117.

O'Malley, P. J. J., R. Babbush, I. D. Kivlichan, J. Romero, J. R. McClean, R. Barends, J. Kelly, P. Roushan, A. Tranter, N. Ding, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, A. G. Fowler, E. Jeffrey, A. Megrant, J. Y. Mutus, C. Neill, C. Quintana, D. Sank, A. Vainsencher, J. Wenner, T. C. White, P. V. Coveney, P. J. Love, H. Neven, A. Aspuru-Guzik, and J. M. Martinis (2016). "Scalable Quantum Simulation of Molecular Energies". *Phys. Rev. X* 6:031007.

Ouyang, Q., P. D. Kaplan, S. Liu, and A. Libchaber (1997). "DNA solution of the maximal clique problem". *Science* 278:446–449.

Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley.

Peruzzo, A., J. McClean, P. Shadbolt, M-H Yung, X-Q Zhou, P.J. Love, A. Aspuru-Guzik, and J.L. O'Brien (2014). "A variational eigenvalue solver on a photonic quantum processor". *Nature Comm.* 5:4213.

Potgieter, P. H. (2006). "Zeno machines and hypercomputation". *Theoretical Computer Science* 358(1):23–33.

Qian, Lulu, David Soloveichik, and Erik Winfree (2010). “Efficient Turing-Universal Computation with DNA Polymers”. *DNA Computing and Molecular Programming.* Vol. 6518. LNCS. Springer, pp. 123–140.

Rado, Tibor (1962). “On non-computable functions”. *The Bell System Technical Journal* 41(3):877–884.

Reif, J., J. Tygar, and Y. Akitoshi (1994). “Computability and complexity of ray tracing”. *Discrete and Computational Geometry* 11(3):265–288.

Reiher, Markus, Nathan Wiebe, Krysta M. Svore, Dave Wecker, and Matthias Troyer (2017). “Elucidating Reaction Mechanisms on Quantum Computers”. 114(29):7555–7560.

Rønnow, T. F., Z. Wang, J. Job, S. Boixo, S. V. Isakov, D. Wecker, J. M. Martinis, D. A. Lidar, and M. Troyer (2014). “Defining and detecting quantum speedup”. *Science* 334(6195):420–424.

Salathe, Y., M. Mondal, M. Oppliger, J. Heinsoo, P. Kurpiers, A. Potocnik, A. Mezzacapo, U. Las Heras, L. Lamata, E. Solano, S. Filipp, and A. Wallraff (2015). “Digital quantum simulation of fermionic models with a superconducting circuit”. *Phys. Rev. X* 5:021027.

Saunders, Daniel (2016). “A Survey and Discussion of Memcomputing Machines”. djsaunde.github.io/survey-discussion-memcomputing.pdf.

Schaul, Tom, Julian Togelius, and Jürgen Schmidhuber (2011). *Measuring intelligence through games.* eprint: arXiv:1109.1314[cs.AI].

Schuch, N. and F. Verstraete (2009). “Computational complexity of interacting electrons and fundamental limitations of density functional theory”. *Nature Physics* 5:732–735.

Schuld, M., I. Sinayskiy, and F. Petruccione (2015). “An introduction to quantum machine learning”. *Contemporary Physics* 56(2):172–185.

Seelig, G. and D. Soloveichik (2009). “Time-complexity of multilayered DNA strand displacement circuits”. *DNA Computing and Molecular Programming.* Ed. by R. Deaton and A. Suyama. Vol. 5877. LNCS. Springer, pp. 144–153.

Shapiro, Ehud (2012). “A mechanical Turing machine: blueprint for a biomolecular computer”. *Interface focus* 2:497–503.

Shor, P. W. (1997). “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer”. *SIAM Journal of Computing* 26:1484–1509.

Siegelmann, H. (1995). “Computation beyond the Turing limit”. *Science* 268(5210):545–548.

Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis (2016). “Mastering the game of Go with deep neural networks and tree search”. *Nature* 529(7587):484–489.

Sipser, Michael (1997). *Introduction to the Theory of Computation.* PWS Publishing.

Stepney, Susan (2009). “Non-Classical Hypercomputation”. *International Journal of Unconventional Computing* 5(3–4):267–276.

Traversa, Fabio L. and Massimiliano Di Ventra (2015). “Universal Memcomputing Machines”. *IEEE transactions on neural networks and learning systems* 26(11):2702–2715.

Traversa, Fabio L. and Massimiliano Di Ventra (2017). “Polynomial-time solution of prime factorization and NP-complete problems with digital memcomputing machines”. *Chaos* 27(2):023107.

Traversa, Fabio L., C. Ramella, F. Bonani, and Massimiliano Di Ventra (2015). “Memcomputing NP-complete problems in polynomial time using polynomial resources and collective states”. *Science Advances* 1(6) :e1500031.

Turing, Alan M. (1937). “On Computable Numbers, with an Application to the Entscheidungsproblem”. *Proc. London Mathematical Society* s2-42(1) :230–265.

Unger, R. and J. Moult (1993). “Finding the lowest free energy conformation of a protein is an NP-hard problem: Proof and implications”. *Bulletin of Mathematical Biology* 55(6):1183–1198.

Valiron, B., N. J. Ross, P. Selinger, D. S. Alexander, and J. M. Smith (2015). “Programming the Quantum Future”. *Communications of the ACM* 58(8) :52–61.

Vergis, A., K. Steiglitz, and B. Dickinson (1986). “The complexity of analog computation”. *Mathematics and Computers in Simulation* 28(2):91–113.

Viglietta, G. (2012). “Gaming Is a Hard Job, But Someone Has to Do It!” *Fun with Algorithms.* Ed. by E. Kranakis, D. Krizanc, and F. Luccio. Vol. 7288. LNCS. Springer, pp. 357–367.

Wagon, Stan (1985). *The Banach–Tarski Paradox.* Cambridge University Press.

Wang, Hao (1961). “Proving theorems by pattern recognition–II”. *Bell System Technical Journal* 40(1):1–41.

Wang, Y., F. Dolde, J. Biamonte, R. Babbush, V. Bergholm, S. Yang, I. Jakobi, P. Neumann, A. Aspuru-Guzik, J.D. Whitfield, and J. Wrachtrup (2015). “Quantum simulation of helium hydride cation in a solid-state spin register”. *ACS Nano* 9:7769.

Wangersky, Peter J. (1978). “Lotka-Volterra Population Models”. *Annual Review of Ecology and Systematics* 9:189–218.

Wapner, Leonard M. (2005). *The Pea and the Sun: a mathematical paradox.* A K Peters.

Watrous, J. (n.d.). “Quantum computational complexity”. *Encyclopedia of Complexity and Systems Science.* Ed. by R. A. Meyers. Springer, pp. 7174–7201.

Wecker, Dave and Krysta M. Svore (2014). *LIQUi| >: A Software Design Architecture and Domain-Specific Language for Quantum Computing.* eprint: arXiv:1402.4467[quant-ph].

Wendin, G. (2017). "Quantum information processing with superconducting circuits: a review". *Reports on Progress in Physics* 80:106001.

Whitfield, J. D., P. J. Love, and A. Aspuru-Guzik (2013). "Computational complexity in electronic structure". *Physical Chemistry Chemical Physics* 15:397–411.

Wiebe, Nathan, Ashish Kapoor, and Krysta M. Svore (2014). *Quantum Deep Learning.* eprint: arXiv:1412.3489[quant-ph].

Wiebe, Nathan, Ashish Kapoor, and Krysta M. Svore (2015). "Quantum algorithms for nearest-neighbor methods for supervised and unsupervised learning". *Quantum Information and Computation* 15(3-4):316–356.

Wittek, Peter (2016). *Quantum Machine Learning: What Quantum Computing Means to Data Mining.* Academic Press.

Wolpert, David H. (2012). *What the no free lunch theorems really mean; how to improve search algorithms.* Working Paper 2012-10-017. SFI.

Wolpert, David H. and William G. Macready (1997). "No Free Lunch Theorems for Optimization". *IEEE Trans. Evol. Comp* 1(1):67–82.

Woods, Damien and Thomas J. Naughton (2005). "An optical model of computation". *Theoretical Computer Science* 334:227–258.

Woods, Damien and Thomas J. Naughton (2009). "Optical computing". *Applied Mathematics and Computation* 215(4):1417–1430.

Wu, Kan, Javier García de Abajo, Cesare Soci, Perry Ping Shum, and Nikolay I Zheludev (2014). "An optical fiber network oracle for NP-complete problems". *Light: Science & Applications* 3(2):e147.

Yang, J. C.-C. (2013). "Computational complexity and decidability of tileability". PhD thesis. UCLA. URL: www-users.math.umn.edu/~jedyang/papers/yang-thesis.pdf.

Zintchenko, I., E. Brown, and M. Troyer (2015). "Recent developments in quantum annealing". www.scottaaronson.com/troyer.pdf.