mark.sprevak@ed.ac.uk

Not all computations are effective methods

Mark Sprevak
University of Edinburgh

9 May 2022

An effective method is a computational method that might, in principle, be executed by a human. In this paper, I argue that there are non-hypercomputational methods for computing that are not effective methods. The examples I consider are taken primarily from quantum computing, but these are only meant to be illustrative of a much wider class. Quantum inference and quantum parallelism involve abstract steps that might be implemented in multiple physical systems, but cannot be implemented, or at least not at will, in an idealised human. Recognising that not all computational methods are effective methods is important for at least two reasons. First, it is needed to correctly state the results of Turing and other founders of computation theory. Turing is sometimes said to have offered a 'replacement' for the notion of an effective method with his notion of a Turing machine. I argue that such a replacement view only holds under limited circumstances. Second, not distinguishing the two notions can lead to mistakes when quantifying over the class of all possible computational methods. Such quantification is common in philosophy of mind in thought experiments that explore the limits of computational functionalism. I argue that these 'homuncular' thought experiments should not be treated as valid.

1 Introduction

What is the relationship between the notion of a computation and that of an effective method? A number of authors assume that the two notions are coextensive. Indeed, some treat the terms 'effective method' and 'computation' as not just extensional equivalents but also as synonyms. The claim made by this paper is that any such equation is false: not all computations are effective methods.

Distinguishing effective methods from computational methods is important for a number of reasons. First, it is needed to accurately represent the historical motivations of the founders of computation theory, such as Turing, and to correctly state their results in a modern context. Second, not distinguishing between the two has the potential to adversely affect our reasoning when we quantify over the class of possible computational methods. For example, if one thinks that some mental processes are computational processes, one might be led to an incorrect view about the possible nature of those processes – that they must the kinds of things that 'little men' might take over. While this kind homuncular thinking might serve as a rough heuristic or explanatory device when first introducing computational ideas about the mind, it is simply not the right way to understand computational processes. Not all computational methods are human executable, even in principle.

Turing famously developed a formal predicate that aimed to make the informal idea of an effective method more precise. This formalisation, the Turing machine, is sometimes described as offering a 'definition', an 'analysis', or a 'replacement' for that informal notion. I argue that care should be taken in interpreting these claims. Turing's formalisation may serve as an adequate replacement for the informal notion in certain contexts, but not in every context. In particular, if one chooses to individuate computational methods in a way that preserves differences between methods that compute the same function – for example, if one is a functionalist about the mind or one cares about the complexity profiles of different methods for computing the same function – then the methods available to a system that uses effective methods cannot be identified with those available to a system that uses any computational method.

The argument of this paper runs as follows. In Section 2, I distinguish my argument from two related arguments in the literature: the first is that hypercomputers provide examples of non-effective computation; the second is that all computations should be individuated extensionally (by their overall input–output profile). In Section 3, I analyse the notion of an effective method; I argue that a core requirement is that an effective method should, in principle, be human executable. In Section 4, I consider the objection that Turing offered a precisification of the concept of an effective method that would allow us to dispense with the informal notion. In Section 5, I examine instances of the claim that all computations are effective methods and explore some of their damaging consequences. In Section 6, I describe two examples, taken from quantum computing, of computational methods that are not effective methods. In Section 7, I consider the objection that quantum computing methods may still be executed by hand if a human were to simulate step-by-step the evolution of the wave function.

2 Distinguishing features of this argument

The argument in this paper should be distinguished from similar arguments in the literature that (i) depend on hypercomputation; or (ii) concern differences in computational functions rather than in computational methods.

2.1 No dependence on hypercomputation

Hypercomputers are hypothetical real or notional systems that compute functions that cannot be computed by any effective method.[1] Hypercomputers generally deploy some overtly 'non-effective' element as part of their design – some special extra resource that is not available to a human working by themselves. The exact nature of this resource may vary between different machines. It might, for example, take the form of being able to take an infinite number of steps in finite time, of being able to store or manipulate real numbers with infinite precision, or having an 'oracle' subunit that provides the machine the answer to a problem via some unspecified means.[2]

It might seem natural to turn first to hypercomputers to justify a claim that not all computations are effective methods. Shagrir and Pitowsky develop an argument along these lines. After introducing various hypercomputer designs, they write:

> 'effective computation' (i.e., calculation by means of effective procedures) encompasses a wide, and an important, class of computations, but not necessarily all computations … none of the hyper-machines described in the literature computes by means of effective procedures. (Shagrir and Pitowsky, 2003, p. 94)

If one accepts that hypercomputers are legitimate computers and the methods they employ are representative of computational methods, then it appears that no more needs to be said. Not all computational methods are effective methods because the computational methods used by hypercomputers are (clearly) not effective methods.

Complicating this dialectical use of hypercomputers, however, are two issues.

First, the hypercomputers that have been proposed to date only exist as notional entities. It is unclear whether they correspond to possibilities that are in any reasonable sense accessible to us. It is unknown whether the kinds of non-effective resources required by hypercomputers, needed for them to do super-Turing computation, could be physically implemented in our universe, and even if they could,

[1]See Copeland and Proudfoot (1999).

[2]For examples of proposed hypercomputers, see Copeland and Sylvan (1999); Copeland (2002); Copeland (2004); Syropoulos (2008).

whether they could be exploited to perform computations in a practical way.[3] This may prompt one to wonder whether we should treat hypercomputers as exactly on a par with – as representative samples of – the class of computational methods. Notwithstanding the existence of notional hypercomputers, perhaps all computational methods that *can be implemented*, or *implemented in some practicable way*, are effective methods. If one's primary interest is in computational methods that can actually be physically implemented – for example, the computational methods that might be implemented in the brain – then perhaps one can ignore or bracket off considerations about non-effectiveness specific to hypercomputers.

Second, even if one ignores issues about the implementation or use of hypercomputers, it is common for both advocates and critics of hypercomputers to characterise hypercomputers as not computing in the full or ordinary sense of the term. Németi and Dávid (2006) talk of their machines using computational methods in a 'broad' or 'extended' sense. Copeland (1997) describes them as satisfying a 'nonclassical' conception of computation. Turing (1939) refers to instances of hypercomputation as 'relativised' computation: computation relative to the assumption that some problem that is uncomputable in the ordinary sense has been solved. These characterisations all suggest that a distinction is to be drawn between an ordinary concept of computation and an extended/relativised concept that pertains to hypercomputers. As with the previous point about implementation, this threatens to deaden the force of the claimed result. Not all (hyper)computations are effective methods, but perhaps all *ordinary* computations are.

This paper deliberately avoids appeal to hypercomputers to justify the claim that not all computations are effective methods. This is not to suggest that either of the two worries above hold, but only that one does not need to go as far as super-Turing computation in order to establish the relevant claim.

The examples I use to justify the claim are taken from quantum computing. These have been chosen because (i) they are known to be physically implementable (and indeed are already physically implemented and practically used); and (ii) they are widely regarded as computing in the full ordinary (non-hyper, non-extended) sense.

In their seminal paper on hypercomputation, Copeland and Sylvan wrote:

> It is perhaps surprising that not all classical algorithms are manual methods. That this is in fact the case has emerged from recent work on quantum computation … Algorithms for quantum Turing machines are not in general manual methods, since not all the primitive operations made available by the quantum hardware can be performed by a person unaided by machinery. (Copeland and Sylvan, 1999, p. 55)

[3]For a range of objections along these lines, see Button (2009); Davis (2004); Piccinini (2011).

After making this observation, they immediately turn to consider hypercomputation (non-classical algorithms). They do not return to or explore non-hypercomputational methods (classical algorithms) that are not effective (manual) methods. This paper could be understood as an attempt to expand on and defend Copeland and Sylvan's original observation.

2.2 Computations should be individuated their internal workings

It is common for discussions of effective methods to focus on the question of which *functions* are computable.[4] In that context, computational methods are normally individuated *extensionally*: by their input–output behaviour, the set of ordered pairs of their possible inputs and outputs.[5] My focus in this paper is not on this question about computability, but on questions about which methods are used for computing a function. None of the examples I consider involve computation of a function that cannot also be computed by an effective method. Rather, the question is whether the deployment of a *computational method* entails the deployment of an *effective method*. This is not a question about computability as such; it is a question about the conditions under which effective methods are and are not instantiated in a computational system. In order to be able to ask this question coherently, and to distinguish it from the standard question about computability, it is important that we do not individuate computational methods in a purely extensional fashion. To this end, in the context of this paper, computational methods are individuated by their internal workings.

It should be stressed that this shift in how we individuate is not ad hoc or unmotivated. The standard question about extensional equivalence is important, but internal differences between methods that compute the same function matter too. These differences are relevant to proposals about computational functionalism regarding the mind. According to such views, what is required for having a mental life is not only having the right behavioural responses – computing the right input–output function – but also the specific method by which that behaviour is generated. If one wishes to reproduce or model cognition in an artificial system, then reproducing or modelling that computational method, and not merely its input–output results, is essential.[6]

Internal differences between computational methods also matter to computer science. Different methods for computing the same function might impose significantly different demands on resource usage, rendering some methods more or less feasible

[4]Or equivalently, which numbers are computable (Sect 4), or which puzzles can be solved by computational means (Sect 5).

[5]In the terms of Church (1941), by their function-in-extension.

[6]See Block (1981) for a classic discussion of this problem.

to implement. Measures of that resource usage – normally summarised by a function that bounds how much time or space a method uses in the worst case – are of considerable theoretical and practical interest in computer science.

BubbleSort and MergeSort are widely regarded as paradigmatic examples of computational methods that compute the same function in different ways. BubbleSort and MergeSort both map an unordered list of elements to a sorted list of the same elements. BubbleSort works by swapping pairs of adjacent elements in-place until the entire list is sorted. MergeSort works by splitting a list to create sublists which it then recursively merges to produce a final sorted version. BubbleSort can be shown to have a run-time complexity of $O(n^2)$ and space complexity of $O(1)$, whereas MergeSort has a run-time complexity of $O(n \log n)$ and space complexity of $O(n)$.[7] A powerful motivation for drawing distinction between these two computational methods – for treating them as two methods rather than one – is that they place different demands on the resources of any system that implements them. This principle – that different complexity profiles indicate different computational methods – will be important later in this paper.

Complexity profiles are not the only thing that matter when individuating computational methods. Variants of either BubbleSort or MergeSort might share the same complexity profile but still be distinguished as 'different'. One might imagine introducing a range of variations from minor (e.g. extra debugging checks) to major (e.g. approaching the problem in a completely new way) into the sequence of operations of any computational method. At which point does a variation in this sequence result in a new computational method? Which factors – above and beyond differences in complexity profile – matter when individuating methods?

Currently, there is no agreed answer to this question, or at least none that takes the form of an exhaustive set of necessary and sufficient conditions. It is a hard question to address with an explicit theory.[8] There are many 'borderline' cases of identity of computational methods and the standards for what counts as the 'same' might vary depending on context and what is currently of interest.[9] Notwithstanding the existence of understandably 'hard' cases for a theory of algorithm individuation however, there are also plenty of 'easy' cases too. BubbleSort and MergeSort are regarded as *different* computational methods for sorting a list, and classified as such both clearly and relative to any interests. As remarked above, a powerful

[7] In this notation, n is the size of the list and $O(g(n))$ provides an asymptotic upper bound on the resource consumption: for large enough n, resource consumption is always less than or equal to some constant times the $g(n)$ function named inside the $O(\cdot)$. For more on complexity theory and use of big-O notation to measure resource usage, see Papadimitriou (1994).

[8] However, see Dean (2016) for a review of contemporary analytic approaches to this problem, including those of Gurevich (1999; 2000) and Moschovakis (2001).

[9] For a helpful analysis of these problems, see Blass, Dershowitz and Gurevich (2009).

consideration in this context – one that renders this judgement an easy one – is their difference in complexity profile. In computer science, it is simply unheard-of for two methods with different complexity profiles to be classified as the 'same' for any purposes other than extensional equivalence.[10] My claim is that, relative to this widely accepted, clear, and robust standard for individuation, there are computational methods that are not effective methods.

3 What is an effective method?

Copeland provides one of the best characterisations of an effective method:

> A method, or procedure, *M*, for achieving some desired result is called 'effective' (or 'systematic' or 'mechanical') just in case:
>
> 1. *M* is set out in terms of a finite number of exact instructions (each instruction being expressed by means of a finite number of symbols);
> 2. *M* will, if carried out without error, produce the desired result in a finite number of steps;
> 3. *M* can (in practice or in principle) be carried out by a human being unaided by any machinery except paper and pencil;
> 4. *M* demands no insight, intuition, or ingenuity, on the part of the human being carrying out the method. (Copeland, 2020)

Or more briefly:

> A mathematical method is termed 'effective' or 'mechanical' if and only if it can be set out in the form of a list of instructions able to be followed by an obedient human clerk … who works with paper and pencil, reliably but without insight or ingenuity, for as long as is necessary. (Copeland, 2000, p. 12)

What he says is consistent with a wide range of historical and contemporary sources:

> Turing examined however *human* mechanical computability and exploited, in sharp contrast to Post, limitations of the human computing

[10]See Knuth (1981), p. 97 who suggests that a distinguishing feature of computer science is that algorithms should be individuated by their complexity class. He argues that this 'algorithmic' mode of thinking separates the thought processes of earlier mathematicians from those of later computer scientists (pp. 96–98). See Dean (2016), pp. 20–29; Shagrir (2016) for further discussion of how complexity profiles matter to the individuation of computational methods.

> agent to motivate restrictive conditions … Turing asked in the historical context in which he found himself *the* pertinent question, namely, what are the possible processes a human being can carry out (when computing a number or, equivalently, determining algorithmically the value of a number theoretic function)? (Sieg, 2002, p. 395)

> [computable problems are those] which can be solved by human clerical labour, working to fixed rule, and without understanding (Turing, 1992, pp. 38–39)

> [With regard to what is effectively calculable] Both Church and Turing had in mind calculation by an abstract human being using some mechanical aids (such as paper and pencil). (Gandy, 1980, p. 123)

> *Turing's analysis makes no reference whatsoever to calculating machines.* Turing machines appear as a result, as a codification, of his analysis of calculation by humans [previously defined as 'effective calculability'] (Gandy, 1988, p. 77)

> Roughly speaking, an algorithm [previously defined as an 'effective procedure'] is a clerical (i.e., deterministic, book-keeping) procedure which can be applied to any of a certain class of symbolic *inputs* and which will eventually yield, for each such input a corresponding symbolic *output*. (Rogers, 1967, p. 1)

> *Effectiveness*. An algorithm is also generally expected to be *effective*, in the sense that its operations must all be sufficiently basic that they can in principle be done exactly and in a finite length of time by someone using pencil and paper. (Knuth, 1997, p. 6)

> [an effective procedure is] a list of instructions … that in principle make it possible to determine the value $f(n)$ for any argument n … The instructions must be completely definite and explicit. They should tell you at each step what to do, not tell you to go ask someone else what to do, or to figure out for yourself what to do: the instructions should require no external sources of information, and should require no ingenuity to execute … (Boolos, Burgess and Jeffrey, 2007, p. 23)

Common to all these suggestions is the idea that an effective method should be capable of being executed *by a lone human being, unaided by any resources except paper and pencil.* The human is allowed an unlimited amount of time, they are assumed not to make errors or get bored, and they have an unlimited supply of paper and pencils. An effective method is a method that can be implemented by such an idealised human worker. Correspondingly, the kinds of operations that can be executed by this kind of idealised human set limits on the class of effective methods.

Some authors have proposed revisionist accounts of 'effective method'. Cleland (2002; 2004) argues that an effective method is a 'quotidian' procedure that has essentially physical, causal consequences, such baking a cake or assembling a child's bicycle. Although a human might follow an effective method, human executability is not a necessary condition – a non-living particle travelling through a vacuum might follow an effective method that no human could replicate. Etesi and Németi (2002) suggest that 'effective method' should refer to any method that can be realised in any physical system, whether that system is an idealised human being or not. Shagrir (2002) argues that the term 'effective method' has undergone a meaning shift: in 1936, 'effective method' meant a method that was in principle human executable, but today it means any symbolic operation that makes use of a finite procedure, and correspondingly that it may refer to methods executable by humans, physical systems, or abstract automata.[11]

Shapiro (2006) provides a helpful discussion of how our various different ideas about 'effectiveness' might have been sharpened in many competing ways. He argues that the notion of effectiveness exhibited 'open texture', meaning that the full range of possible cases to which it correctly applies and does not apply was not entirely pinned down by its semantic content. Shapiro's point, however, pertains primarily to the history and development of the concept of effectiveness: our early, relatively inchoate ideas *could* have been sharpened in different ways. He does not argue that *today* in mathematics we are free to adopt different conceptions, or that adopting an alternative, e.g. non-human, conception of what an effective method is would be equally good for the purposes of doing computer science or mathematical logic – indeed, he is clear that this is not the case. In line with others working on the foundations of computing, he suggests that the idealisation described above – a human working with unbounded time and computing space – is a way of signalling that one is talking about a particular subtype of procedure, one that has important connections to pre-existing ideas about mathematical provability, decidability, and verifiability. Irrespective of the rise of non-human machines or shifting interests within computer science, there is still a need to refer to this subset of methods.[12]

[11]Some critics of Turing argued that his human-centric characterisation of an effective method was not too narrow, but too broad. They argued that the definition should be narrowed by adding a requirement that the number of steps taken by the human clerk be determinable or bounded in advance. For critical discussion of these proposals, see Gandy (1988); pp. 59–60; Mendelson (1963), p. 202; Rogers (1967), p. 5.

[12]For further defence of the human-centric construal of 'effective method', see Black (2000); Button (2009); Copeland (2020); Gandy (1988); Smith (2013).

4 Didn't Turing define 'effective method'?

In textbooks on mathematical computation theory, talk of effective methods often disappears once an appropriate formal predicate like 'Turing machine' has been introduced. This is often glossed by saying that Alan Turing provided a formal *definition*, *analysis*, or *replacement* for the informal concept of an effective method. Thanks to Turing, we can dispense with the informal notion and work instead with a more formal, precise concept. A small wrinkle is that there is more than one 'definition' of effective method available – Church introduced an alternative formalism with the λ-calculus, Gödel introduced another with general recursive functions, and there are many others. However, because all known formalisms can be shown to be 'equivalent' to one another, the choice between them can be regarded as largely a matter of convenience. Thus, talk of the human-centric notion of 'effective method' can be replaced with the formal, precise notion of 'Turing machine' (or equivalent):

> Turing's work is a paradigm of philosophical analysis: it shows that what appears to be a vague intuitive notion has in fact a unique meaning which can be stated with complete precision. (Gandy, 1988, p. 79)

> Church's thesis is the proposal to identify an intuitive notion with a precise, formal, definition (Folina, 1998, p. 311)

> In 1928, the notion of an algorithm [effective method] was pretty vague. Up to that point, algorithms were often carried out by human beings using paper and pencil … Attacking Hilbert's problem forced Turing to make precise exactly what was meant by an algorithm. To do this, Turing described what we now call a *Turing machine* (Matuschak and Nielsen, 2019)

> If Turing's thesis is correct, then talk about the existence and non-existence of effective methods can be replaced throughout mathematics, logic and computer science by talk about the existence or non-existence of Turing machine programs. (Copeland, 2020)

Turing himself, perhaps in a relatively unguarded moment, appears to endorse this:

> … one can reduce it [the definition of a solvable puzzle] to the definition of 'computable function' or 'systematic [effective] procedure'. A definition of any one of these would define all the rest. Since 1935 a number of definitions have been given [Turing machines, the λ-calculus, the μ-recursive functions, etc.], explaining in detail the meaning of one or other of these terms, and these have all been proved equivalent to one another (Turing, 2004/1954, p. 589)

I call this the 'replacement theory' of effective methods. If the replacement theory is correct, then the concept of an effective method can be replaced by the notion of a Turing machine (or equivalent). The question the paper addresses could then be rephrased as a question about which computational systems can and cannot instantiate Turing machines (or equivalent formalism).

It is important to see this is not the case. The problem is not that the replacement theory is false or unwarranted, but that it only holds – and it was only justified by Turing – under certain limited circumstances. To set an upper bound regarding those circumstances, it should be evident that an entirely unrestricted version of the replacement theory would be implausible. The concept of an effective method and the concept of a Turing machine are not identical concepts, they do not have identical semantic content – if they did, it would have taken no insight on Turing's part to establish a connection between them. In order to make sense of Turing's achievements here, we need to rule out that 'effective method' is an unqualified synonym for 'Turing machine'. The key question then is when it is, and is not, legitimate to replace one notion with the other.

In Section 9 of Turing's (1936) paper, he says that his goal is to show that a human working by hand and his formal machine (a Turing machine) compute the same numbers (the title of the section is 'The extent of the computable numbers'). If this relationship holds, then a certain kind of intersubstitutability between the terms would be warranted. Provided one's concern is *only* to identify which numbers are computable, then talk of effective methods could be freely replaced with that of Turing machines (or an equivalent formalism). For replacing one term with the other would have no effect on the validity of one's reasoning about the extent of the numbers that can be computed.

One of the key arguments that Turing gives to justify this is to say that his machine and a human clerk go through an *analogous* process when they compute a number. He does not, however, suggest that they go through an *identical* process, or that the operations of a Turing machine include all and only those that a human worker could take. Rather, he claims that the *results* the human worker obtains without insight or ingenuity are also reproducible by an appropriate series of steps of a Turing machine. In other words, in this section, Turing does not claim that effective methods *are* Turing machines, but that the set of numbers that can be computed by an effective method are the same as the set of numbers that can be computed by a Turing machine. If one's sole concern is to determine the membership of that set, then talk of effective methods can be replaced with that of Turing machines (or another extensionally equivalent formalism).

It is worth noting that Turing did not argue – he did not need to argue – that the methods useable by a Turing machine are identical to the methods useable by a clerk.

Indeed, such a claim would be almost certainly false, and for reasons independent of the main argument of this paper. The steps and operations of a Turing machine – the basic operations that change the state of the head and that make marks on the tape – are not the only ways for a human or any other system to effectively calculate a number. The alternative models of Church, Gödel, and others show that there are other ways to effectively calculate that do not involve those basic operations. The sequence of their basic operations might, for example, involve reduction operations in the λ-calculus, or minimisation and recursion operations over the μ-recursive functions. It is a palpable truth to anyone working in computer science that different computational formalisms support different types of computational method, and that 'porting' methods between computational formalisms is often non-trivial. One cannot take a computational method that runs on a Turing machine and run *exactly the same method*, without changes, on a system that operates according to the rules of the λ-calculus. One might attempt to create an analogous process – one with different internal characteristics expressed in terms of the basic operations and idioms of the λ-calculus – that computes the same functions in a roughly similar manner. The methods of the λ-calculus are not identical to those of a Turing machine, even if both are effective methods. A human clerk might follow any one of these effective methods when computing a number. Hence, Turing machines in this context cannot be identified with effective methods.

5 All computations are effective methods

Here are some examples of the claim that all computations are effective methods:

> An algorithm or effective method … is a procedure for correctly calculating the values of a function or solving a class of problems that can be executed in a finite time and mechanically – that is, without the exercise of intelligence or ingenuity or creativity … A computation is anything that … calculates the values of a function or solves a problem by following an algorithm or effective method. (Burkholder, 2000, p. 47)

> The logician Turing proposed (and solved) the problem of giving a characterization of *computing machines* in the widest sense – mechanisms for solving problems by effective series of logical operations. (Oppenheim and Putnam, 1958, p. 19)

> We have assumed the reader's understanding of the general notion of effectiveness, and indeed it must be considered as an informally familiar mathematical notion, since it is involved in mathematical problems of a frequently occurring kind, namely, problems to find a method of

> computation, i.e., a method by which to determine a number, or other thing, effectively. We shall not try to give here a rigorous definition of effectiveness, the informal notion being sufficient to enable us, in the cases we shall meet, to distinguish methods as effective or non-effective … The notion of effectiveness may also be described by saying that an effective method of computation, or algorithm, is one for which it would be possible to build a computing machine (Church, 1956, p. 52)

> Sometimes computers are called information processors … *How* they process or manipulate is by carrying out effective procedures … Computation [means] the use of an algorithm … also called an 'effective method' or a 'mechanical procedure' … to calculate the value of a function. (Crane, 2003, pp. 102, 233)

> The functional organisation of mental processes can be characterized in terms of effective procedures, since the mind's ability to construct working models is a computational process. (Johnson-Laird, 1983, pp. 9–10)

> … [a] procedure admissible as an 'ultimate' procedure in a psychological theory [will fall] well within the intuitive boundaries of the 'computable' or 'effective' as these terms are presumed to be used in Church's Thesis (Dennett, 1978, p. 83)

The quotations above illustrate that the claim has been made in a variety of different contexts. The final three quotations provide examples of how it can constrain thinking about the mind.[13]

Searle's Chinese room argument provides a particularly clear example of the latter phenomenon (Searle, 1980). Searle's argument may be challenged on many points, but among them is the assumption that any computational method can be executed, without loss or distortion, by the human being inside the room who generates Chinese responses. Searle needs a claim like this in order to connect the specifics of his thought experiment (a lone human working inside the room) to his conclusion that no computational method can suffice for understanding. One needs to make an inferential leap from the person inside the room not understanding Chinese

[13]Copeland (1998; 2000) criticises a number the same authors for committing what he calls the 'Church–Turing fallacy'. The fallacy is to assume that any possible physical mechanism could be simulated by some Turing machine. My claim is that the authors make a second mistake in that they assume that any possible computational method is also an effective method. Copeland argues that although 'effective' and 'mechanical' sometimes appear to be synonyms in mathematical logic, the relationship between them should handled with caution. 'Mechanical' should be understood as a term of art and defined in the way described in Section 3. It does not correspond in any straightforward way to our concept of a physical mechanism.

regardless of which computational method *they follow* to the conclusion that *no possible computational method* could be sufficient for Chinese understanding. Searle cites Turing's analysis of computation to support this step.[14] However, as we have seen, the required claim is not attributable to Turing, and as we will see in the next section, it is false.[15]

The identification of computational methods with effective methods is more deep-seated in the philosophical literature than just Searle's argument. It is a premise accepted by both advocates and critics of computational models of cognition. A common argumentative move, when reasoning about a computational model of cognition, is to assume that one can replace the computational system's workings with a human working by rote. One might always, with impunity, substitute a computational method with a human-executable method while preserving its computational and functional identity. This has given rise to a widespread form of 'homuncular thinking' about computational methods.

Fodor (1968) describes how an account of knowledge-how, e.g. of how to tie one's shoes, could be given in computational terms. In the course his discussion, Fodor moves freely between a formulation of that knowledge-how in terms of a computation performed by the brain and in terms of basic steps taken by a 'little man' who reads basic instructions and follows them. The unstated assumption is that whatever computational process underlies knowledge-how, it can be reconceptualised in terms of a series of steps taken by a little man who reads basic instructions and follows them without insight or ingenuity. Of course, Fodor is not suggesting that any little man actually lives inside the head. However, he is suggesting that talk of 'the little man stands as a representative *pro tem* for psychological faculties which mediate the integration of shoe-tying behavior by applying information about how shoes are tied.' (ibid., p. 629). He does not entertain the possibility that such talk might provide a distorted or constrained picture of the computational process. He simply assumes that 'little man' talk can always stand in for 'computation' talk without narrowing down the kind of computational processes that actually govern knowledge-how.

Dennett (1978) defends homuncular functionalism about the mind by formulating it in two ways which, like Fodor, he treats as interchangeable.[16] On one formulation, it is the view that a cognitive capacity can be explained by breaking it down into the action of a series of simpler computational subsystems, which are each

[14]See Searle (1992), p. 202 and Searle (personal correspondence).

[15]See Copeland (1993); Sprevak (2007) for a more detailed analysis of this assumption in the Chinese room argument.

[16]See Lycan (1981) for the name 'homuncular functionalism' and a clear reconstruction of the view.

explained in terms of the action of simpler subsubsystems, and so on, until one reaches subsystems whose basic operations are regarded as simple and not requiring any explanation. This is treated as equivalent to the idea that one can explain the cognitive capacity by breaking it down into the actions of a series of notional 'little men' inside the brain who each work without insight or ingenuity. The unstated assumption is again that, whatever computational processes actually underlie cognition and are actually physically realised in the brain, they may be regarded, without loss or distortion, as a series of steps capable of being executed by a group of little men each working to an effective method.

Block (1978) provides a range of arguments against computational theories of phenomenal consciousness based on intuitions about what a collection of little men can and cannot do. In his 'homunculi-headed' thought experiment, the computation that normally takes place inside a human brain via neuronal activity is reproduced by a sequence of steps taken by a collection of little men each working according to an effective method. Block argues that is implausible that a group of little men would instantiate a new qualitative conscious experience as a group agent, and hence that any purely computational account of conscious experience is similarly unlikely to be true. A crucial premise in Block's argument is that the group of little men could reproduce, without loss or distortion, any computational method allegedly characteristic of conscious experience. Like Fodor and Dennett, Block does not justify this. He simply assumes that every computational method must be human executable.

Why do these authors think that this assumption is unproblematic?

One possible motivation might come from misguided intuitions about multiple realisability. Computational methods are multiply realisable because the kind of description that is needed to characterise a computational method does not inherently tie it to being implemented in just one type of physical medium. When the steps of a Turing machine are described, there is no requirement in that description that a system that implements those steps must be made out of lead or wood or steel. In principle, the same computational method might be implemented with silicon chips, clockwork, or in the behaviour of a human clerk. Computational methods are not tied, by virtue of their specification, to being implemented in one physical medium. However, there is a separate and much stronger claim about multiple realisability that is often associated with computation but which is much less plausible. This claim is that any computational method can be realised, in principle, in *any* physical medium. As Putnam notoriously put it: 'We could be made of Swiss cheese and it wouldn't matter' (Putnam, 1975, p. 291). Or, in the specific case of the human clerk, that any computational method can be implemented by some sequence of actions of that clerk. Unlike the former claim, there is no reason to think that this claim is true. It does not follow from the brute fact that computational methods are multiply realisable: just because a computational method can be implemented in *more than one* physical medium that does not mean that it can be implemented in any medium or in the actions of an idealised human clerk. Different physical media have different and varied causal powers. These causal powers enable them to implement some formal operations but not others. There is no reason to think that an idealised human clerk has the causal powers to implement any possible computational method.[17]

Before closing this section, it is worth saying a few words about the term 'algorithm'. The authors cited above suggest that 'algorithm' is interchangeable with *both* 'effective method' and 'computational method'. My claim is that the terms 'effective method' and 'computational method' should be distinguished. If we distinguish these terms, how should we understand the term 'algorithm'? Some authors suggest that 'algorithm' goes with 'effective method' – it refers to only human-executable methods (Button, 2009; Cutland, 1980; Smith, 2013). Other authors suggest that 'algorithm' refers to the wider class, computational methods – i.e. there might be algorithms that are not human-executable (Copeland, 1997; Copeland and Sylvan, 1999; Gurevich, 2011; Soare, 1999).[18] In this paper, I will follow the latter convention and treat 'algorithm' as picking out the more general class. This will allow us to say that there are quantum computing algorithms, even if there are no quantum computing effective methods. Nothing turns on this however, and the argument of this paper may be rephrased if one prefers to understand the term 'algorithm' differently.[19]

6 Quantum computations that are not effective methods

Quantum computers are able to move from input to output using methods that are not open to any idealised human clerk. A human working by hand may be able to compute the same functions as a quantum computer – they may be able to simulate

[17]See Shagrir (1998) for a helpful analysis and criticism of the stronger claim about the multiple realisability of computation.

[18]See also Blass and Gurevich (2006):

> In fact the notion of algorithm is richer these days than it was in Turing's days. And there are algorithms … not covered directly by Turing's analysis, for example, algorithms that interact with their environments, algorithms whose inputs are abstract structures, and geometric or, more generally, non-discrete algorithms. (p. 31)

[19]It is worth noting that the term 'algorithm' has a long history and semantic associations that predate its current connections with either 'computational method' or 'effective method'. See Chabert et al. (1999); Knuth (1972).

a quantum computer's input–output behaviour – but they are not able to use the same method to do so.

Deutsch, Ekert and Lupacchini (2000) describe a simple quantum computer that uses one of these non-effective methods. The computer uses *quantum interference* to compute the NOT function. The NOT function maps an input of 0 to an output of 1 and an input of 1 to an output of 0. Clearly, there is no question here of computing a function that cannot also be computed by a human by hand. The question is whether the method that the quantum computer uses to calculate NOT cannot also be used by an idealised human clerk.

The proposed quantum computer that calculates NOT is composed of two half-silvered mirrors (mirrors that reflect a photon with 50% probability and allow a photon to pass through with 50% probability). The presence of a photon along one path to a half-silvered mirror denotes an input of 1, the presence of a photon along the other path denotes an input of 0, the presence of a photon along one exit path denotes an output of 1, the presence of a photon along the other exit path denotes an output of 0.

A single half-silvered mirror implements a quantum computational gate that Deutsch calls $\sqrt{\text{NOT}}$. If the input to the gate is 0, then the output is measured as either 0 or 1 with 50% probability; similarly, if the input is 1, the output is measured as either 0 or 1 with 50% probability. Formally, if the input is prepared in quantum state $|0\rangle$ (i.e. 0), then the output occurs in superposition state $(|0\rangle - i\,|1\rangle)/\sqrt{2}$ (which, on measurement, results in a 0 or 1 with 50% probability). If the input is prepared in quantum state $|1\rangle$ (i.e. 1), then the output occurs in superposition state $(-i\,|0\rangle + |1\rangle)/\sqrt{2}$ (which also, on measurement, results in a 0 or 1 with 50% probability).[20]

If two half-silvered mirrors are connected together in series as shown in Figure 1, then the overall system computes the NOT function $(0 \rightarrow 1, 1 \rightarrow 0)$. If one did not know better, one might have guessed that this arrangement of half-silvered mirrors would produce a random output. Individual half-silvered mirrors are randomisers, so chaining two mirrors together should produce equally random results. But due to the rules by which physical superposition states evolve in quantum mechanics, the alternative possibilities interfere with each other, such that an input of 0 to the first mirror always yields and output of 1, and an input of 1 to the first mirror always yields an output of 0. This occurs even if a single photon is sent into the system, a

[20]If a superposition state $\alpha\,|0\rangle + \beta\,|1\rangle$ is measured, then the result is 0 with probability $|\alpha|^2$, and 1 with probability $|\beta|^2$, with $|\alpha|^2 + |\beta|^2 = 1$. A $\sqrt{\text{NOT}}$ gate performs the operation on the quantum state vector $\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$ described by the complex matrix $\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}$.

phenomenon known as single-particle interference.

Formally, this can be seen as follows. The first half-silvered steps maps $|0\rangle \rightarrow (|0\rangle - i\,|1\rangle)/\sqrt{2}$. The second half-silvered mirror applies the same operator to that superposition state, mapping $(|0\rangle - i\,|1\rangle)/\sqrt{2} \rightarrow -i\,|1\rangle$, which, on measurement, results in an output of 1 with 100% probability ($|-i|^2 = 1$). Combining the two transformations, if the input is 0, then the output is 1. Similarly, the first half-silvered mirror maps $|1\rangle \rightarrow (-i\,|0\rangle + |1\rangle)/\sqrt{2}$. The second half-silvered mirror applies the same operator, mapping $(-i\,|0\rangle + |1\rangle)/\sqrt{2} \rightarrow -i\,|0\rangle$, which, on measurement, results in an output of 0 with 100% probability.

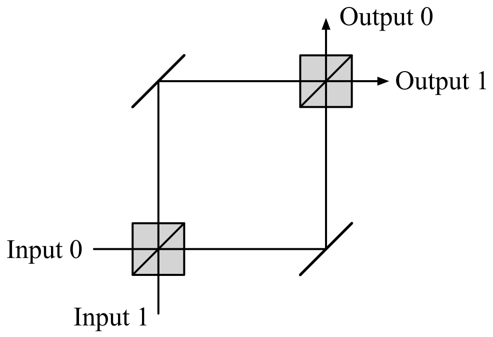

Figure 1: An example of a quantum NOT gate.

The same input–output function, NOT, can of course be calculated by a human, but not using the same method.

It is important to stress that the claim here is not about the specific physical implementation of the quantum computation. The claim is not that the same *photon-and-mirrors* process cannot be reproduced by an unaided human. That is obviously true. The claim is that the same abstract *computational method* cannot be used. There is no suitably equivalent process that a human worker can undergo, even if they are idealised in the way suggested, that calculates input–output in the same way. The method used by the quantum NOT gate is multiply realisable: it might be implemented with photons, electrons, fields, or atomic nuclei. All exhibit interference patterns that might be exploited to compute NOT in this way. But the method cannot be implemented in an unaided human working by hand – or at least, not in a controllable way. The method for calculating NOT is multiply realisable, but it cannot be realised at will in an unaided human.[21]

[21]Cf. Nielsen and Chuang (2010), p. 50: 'it is tempting to dismiss quantum computation as yet another technological fad . . . This is a mistake, since quantum computation is an *abstract paradigm* for information processing that may have many *different* implementations in technology.'

We have just seen one non-effective method which exploits interference to move from input to output. Another example of a non-effective computational method is *quantum parallelism*. Quantum parallelism underlies the speedup of some quantum computers over classical computers.

Quantum parallelism allows multiple values of a function $f(x)$ to be calculated in a single step. In the simplest case, if an arbitrary 1-bit function $f(x)$ is applied to an input superposition state $(|0\rangle + |1\rangle)/\sqrt{2}$, then the resultant state would include $(|0, f(0)\rangle + |1, f(1)\rangle)/\sqrt{2}$. This superposition state contains information about *both* $f(0)$ and $f(1)$, but it was obtained using only a single application of $f(x)$.[22] In a more complex environment, every value of an n-bit $f(x)$ can be calculated in a single application of $f(x)$. If $n+1$ bits are prepared in a superposition state, then a single application of $f(x)$ results in the superposition state $(2^{-n/2}) \sum_x |x, f(x)\rangle$, a state that encodes all values of $f(x)$ simultaneously.[23] Quantum parallelism is an non-effective procedure that allows a system to calculate all values of an arbitrary function in one step. It is not a computing method freely available to a human working by hand.

A well-known limitation on methods that employ quantum parallelism is that only a single value of $f(x)$ can recovered from $(2^{-n/2}) \sum_x |x, f(x)\rangle$ by measurement.[24] This limitation is far from fatal, however, to the idea that all values of $f(x)$ are available for computational purposes. This is because before measurement all manner of computational operations may be performed on the superposition state. These operations may affect different components of the wave function – which represent different values of $f(x)$ – in different ways. For example, certain components of the superposition state can arranged to interfere with one another. These interference relations can be constructive or destructive, amplifying the probability of an outcome, or suppressing it. If correctly arranged, such interference relations can combine to calculate some desired global property of $f(x)$: a property that would have required a conventional computer to explicitly evaluate $f(x)$ for many different values of x individually. This means that although only a single value of $f(x)$ can be recovered directly via measurement, all values of $f(x)$ can contribute to the overall output of a computation, rendering this a unique form of parallel computation.

[22]More accurately, a unitary (reversible) operator U_f is applied to the input, U_f: $|x, y\rangle \rightarrow |x, y \oplus f(x)\rangle$, where $\oplus$ indicates addition modulo 2. U_f is used because there is no guarantee that f itself is unitary, and the evolution of a quantum mechanical system must be governed by unitary operators. This modification does not affect the point above.

[23]See Nielsen and Chuang (2010), pp. 30–32.

[24]Strictly, a pair of values can be recovered, $x, f(x)$. The output is a pair because the evolution of the quantum state is governed by unitary operators (quantum computations must be reversible).

The Deutsch–Jozsa algorithm provides an example of how this might work.[25] Suppose Alice picks a function $f(x) : \{1, \ldots, n\} \to \{0, 1\}$ that is either balanced or constant and keeps it secret. A *constant* function yields the same value for all x; a *balanced* function yields 1 for half of x, and 0 for the other half. Bob can send Alice a number and ask her for the value $f(x)$. Bob's task is to determine, with as few queries as possible, whether Alice's function is constant or balanced. Using quantum computing methods, Bob can solve the problem using *just one* evaluation of $f(x)$. In the classical case, Bob requires at least $2^n/2 + 1$ evaluations in order to solve the problem. According to the quantum method, Alice's $f(x)$ is applied once to a superposition state which is then passed through a series of Hadamard gates. A Hadamard gate is a quantum operator that works in a similar way to Deutsch's gate, but performs a transformation over the real numbers.[26] If Alice's function is balanced, the various components of the superposition state $\sum_x |x, f(x)\rangle$ cancel each other out to yield the answer 0. If Alice's function is constant, the components of the superposition state constructively interfere to yield the answer 1.[27] The full details of the Deutsch–Jozsa algorithm are complex, but the key point to note is that the way in which Bob solves the problem uses a method that requires only a single application of $f(x)$ in a way that is not available to a human working by hand.

The problem that the Deutsch–Jozsa algorithm solves is of little practical interest, but the same kind of method can used in many other ways. Shor's algorithm, for example, uses quantum parallelism to find prime factors in polynomial time (Shor, 1999). It is almost exponentially faster than the most efficient known effective computational method for factoring large numbers (the general number field sieve).[28] Applying the principle from Section 2.2 – that different complexity profiles indicate different computational methods – one can infer that Shor's algorithm is distinct from any computing method known to run on more conventional computing systems (including human clerks).

Quantum parallelism should not be confused with the kind of parallelism commonly found in electronic computers. In a modern electronic computer, multiple computational units may be executed simultaneously to compute more than one value of $f(x)$ within a single time step. In a machine that uses quantum parallelism, a single computational unit is executed *once* to evaluate all values of $f(x)$. Quantum

[25] See Deutsch and Jozsa (1992); Cleve et al. (1998). A simplified version of the algorithm was first described in Deutsch (1985).

[26] The operator provided by a Hadamard gate is given by the real-valued matrix $\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$. Like Deutsch's $\sqrt{\text{NOT}}$, a Hadamard gate may be physically implemented with half-silvered mirrors (Barz, 2015).

[27] See Nielsen and Chuang (2010), pp. 32–36 for the details of the algorithm.

[28] ibid.

parallelism should also be distinguished from the kind of quasi-parallelism found in non-deterministic computers. It would be a mistake to identify a quantum computer that uses the superposition state $(|0, f(0)\rangle + |1, f(1)\rangle)/\sqrt{2}$ with a non-deterministic computer that yields $f(0)$ with 50% probability and $f(1)$ with 50% probability. For a non-deterministic computer, the two alternatives $f(0)$ and $f(1)$ necessarily exclude each other – the system either computes $f(0)$ or $f(1)$ on any given run. For the quantum computer, the two alternatives can interfere with each other to create an output that reflects a global property of $f(x)$.

Although quantum parallelism is implemented in a non-human physical system, just as with interference, what is at issue is not a *photon-and-mirrors* process. Quantum parallelism is multiple realisable: it might be implemented in a physical system with photons, electrons, or atomic nuclei. The claim is not that a human clerk is *different* from any of those physical systems. That is obviously true. Rather, the claim is that photons, electrons, atomic nuclei, etc. are able to employ abstract computational methods that are not available to a human worker.

Nielsen and Chuang revealingly observe that it is *hard* for us to come up with computing methods like Shor's algorithm or the Deutsch–Jozsa algorithm. They explain this by suggesting that humans are cognitively biased towards thinking in terms of effective methods. When we try to think up a computational method to solve a problem, we tend to imagine sequences of steps that would be executable by a human (an example of the homuncular thinking described in the previous section):

> To design good quantum algorithms one must "turn off" one's classical intuition for at least part of the design process, using truly quantum effects to achieve the desired algorithmic end. (Nielsen and Chuang, 2010, p. 7)
>
> If we think about problems using our native intuition, then the algorithms which we come up with are going to be classical algorithms. It takes special insights and special tricks to come up with good quantum algorithms (ibid., p. 173)

The two examples of non-effective methods considered in this section – quantum interference and quantum parallelism – are only a sample of possible non-effective computing methods afforded by quantum computation. Other examples might include sequences of operations that exploit entanglement, quantum teleportation, or counterfactual computation.[29] Plausibly, even physical systems that rely on the rules of classical physics might count as employing methods that cannot be reproduced by a human clerk. Possible examples might include the formal methods employed by DNA computers (Adleman, 1994; Lipton, 1995); enzyme-based computers (Barrett, 2005); slime moulds (Adamatzky, 2016); fungi (Adamatzky, 2018); reservoir computers (Tanaka et al., 2019); and optical computers (Wu et al., 2014). Once one is open to the possibility that not every computational method need be human executable, all manner of non-effective computational processes come into view.

7 Simulating the quantum system by hand

Someone might object that the quantum computing methods described in the previous section *are* executable by a human. All a human would need to do is calculate, step-by-step, the evolution of the underlying quantum wave function. This is what we did ourselves in the case of Deutsch's NOT gate: we applied the relevant quantum operator to input superposition states to calculate the output. There is no reason why an idealised human clerk (with unbounded time, paper, etc.) would not be able to perform a similar step-by-step evaluation to simulate the evolution of any quantum computing system.

The correct response to this objection is to say that although computing the system's wave function by hand is an effective method, it is not the same computational method as letting the quantum system evolve by itself. This can be justified by appealing to the principle, described in Section 2.2, that different complexity profiles indicate different computational methods. There, we saw that although the full story about how to individuate computational methods is unclear, a powerful consideration for thinking that two computational methods are different is that they have different complexity profiles. This fact alone provides a sufficient reason for individuating the computational methods.

Feynman (1982) famously showed that simulating the evolution of a quantum system by hand is a computationally intractable problem. This means that a quantum computer undergoing natural evolution of its wave function, and a human simulating it by calculating its wave function by hand, must have qualitatively different complexity profiles. The human will use exponentially more space (or time) than the quantum computer to produce the same overall output. Calculating the evolution

[29]See Ekert and Jozsa (1998) for algorithms that use quantum entanglement, and Bennett et al. (1993), Gottesman and Chuang (1999) for algorithms that use teleportation. Counterfactual computation is a counterintuitive method in which the intermediate steps of the computations do not take place in the actual world (according to measurement), yet the desired output is still produced, for a proposed application see Hosten et al. (2006).

of an n-bit quantum system by hand would require (at least) 2^n classical bits.[30] For a quantum computer with 400 quantum bits (say, consisting of 400 atomic nuclei), an effective method that calculates the wave function by hand would require more bits for storage than there are estimated particles in the universe. Feynman's result applies to any effective method: any computational method an idealised human might adopt for stepping through the evolution of the quantum system will be exponentially less efficient than running the quantum computer itself. Therefore, running the quantum computation is not the same computational method as having a human simulate the evolution of its wave function by hand.

8 Conclusion

In Section 9 of Turing (1936), he wrote:

> The real question at issue is "What are the possible processes which can be carried out in computing a number?" (Turing, 1936, p. 249)

He goes on to answer that question by describing the operations and methods of what has come to be known as a Turing machine. This appears to implicitly identify the *possible processes which can be carried out in computing a number* with the *methods that can be executed by a Turing machine*. Moreover, the machine that Turing considered was implemented in the actions of an idealised human clerk. As Wittgenstein said, 'Turing's … machines are *humans* who calculate.' (Wittgenstein, 1980/1947, §1096).

We have seen that care should be taken in how this identity claim is interpreted. The possible processes that can be carried out in computing a number outrun both (i) those that might be carried out by a Turing machine and (ii) those that might be carried out by an idealised human worker following an effective method. There are processes for computing that are human executable but not identifiable with a Turing machine (e.g. that involve sequences of operations in the λ-calculus, or over the μ-recursive functions) and there are processes for computing that are not executable by a human at all (under the relevant idealisation) but which are executable by certain non-human computing systems (e.g. quantum computers).

Turing correctly ignored these differences between computing methods. His focus was on relationships of exclusively extensional equivalence – of which functions or numbers are computable. If one cares about only this, then these fine-grained differences between different computational methods do not matter. However, if one is interested in differences in internal working between computing methods – which is commonly the focus in philosophy of mind and computer science – then

[30]Nielsen and Chuang (2010), pp. 48, 204–206.

this identification cannot be made. If one care about how a computational method works, then computing methods cannot be identified with effective methods.

Bibliography

Adamatzky, A., ed. (2016). *Advances in Physarum Machines*. Berlin: Springe.

— (2018). “Towards fungal computer”. In: *Journal of the Royal Society Interface Focus*, p. 20180029. DOI: 10.1098/rsfs.2018.0029.

Adleman, L. M. (1994). “Molecular Computation of Solutions to Combinatorial Problems”. In: *Science* 266, pp. 1021–1024.

Barrett, H. C. (2005). “Enzymatic Computation and Cognitive Modularity”. In: *Mind and Language* 20, pp. 259–287.

Barz, S. (2015). “Quantum computing with photons: introduction to the circuit model, the one- way quantum computer, and the fundamental principles of photonic experiments”. In: *Journal of Physics B: Atomic, Molecular and Optical Physics* 48, p. 083001.

Bennett, C. H., G. Brassard, C. Crépeau, R. Jozsa, A. Peres and W. K. Wootters (1993). “Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels”. In: *Physical Review Letters* 70, pp. 1895–1899.

Black, R. (2000). “Proving Church’s Thesis”. In: *Philosophia Mathematica* 8, pp. 244–258.

Blass, A., N. Dershowitz and Y. Gurevich (2009). “When are two algorithms the same?” In: *The Bulletin of Symbolic Logic* 15, pp. 145–168.

Blass, A. and Y. Gurevich (2006). “Algorithms: A quest for absolute definitions”. In: *Church’s thesis after 70 years*. Ed. by A. Olszewski, J. Woleński and R. Janusz. Ontos Verlag, pp. 24–57.

Block, N. (1978). “Troubles with Functionalism”. In: *Perception and Cognition: Issues in the Foundations of Psychology, Minnesota Studies in the Philosophy of Science*. Ed. by C. W. Savage. Vol. 9. Minneapolis: University of Minnesota Press, pp. 261–325.

— (1981). “Psychologism and Behaviorism”. In: *Philosophical Review* 90, pp. 5–43.

Boolos, G., J. P. Burgess and R. C. Jeffrey (2007). *Computability and Logic*. 5th ed. Cambridge: Cambridge University Press.

Burkholder, L. (2000). “Computing”. In: *A Companion to the Philosophy of Science*. Ed. by W. H. Newton-Smith. Oxford: Blackwell, pp. 44–55.

Button, T. (2009). “SAD Computers and two versions of the Church-Turing Thesis”. In: *The British Journal for the Philosophy of Science* 60, pp. 765–792.

Chabert, J.-L., C. Weeks, E. Barbin, J. Borowczyk, M. Guillemot, A. Michel-Pajus, A. Djebbar and J.-C. Martzloff (1999). *A History of Algorithms: From the Pebble to the Microchip*. Berlin: Springer-Verlag.

Church, A. (1941). *The Calculi of Lambda-Conversion*. Princeton, NJ: Princeton University Press.

— (1956). *Introduction to Mathematical Logic*. Princeton, NJ: Princeton University Press.

Cleland, C. E. (2002). “On effective procedures”. In: *Minds and Machines* 12, pp. 159–179.

— (2004). “The concept of computability”. In: *Theoretical Computer Science* 317, pp. 209–225.

Cleve, R., A. Ekert, C. Macchiavello and M. Mosca (1998). “Quantum algorithms revisited”. In: *Proceedings of the Royal Society, Series A* 454, pp. 339–354.

Copeland, B. J. (1993). “The curious case of the Chinese gym”. In: *Synthese* 95, pp. 173–186.

— (1997). “The broad conception of computation”. In: *American Behavioral Scientist* 40, pp. 690–716.

— (1998). “Turing’s O-machines, Searle, Penrose and the brain”. In: *Analysis* 58, pp. 128–138.

— (2000). “Narrow versus wide mechanism”. In: *The Journal of Philosophy* 97, pp. 5–32.

— (2002). “Hypercomputation”. In: *Minds and Machines* 12, pp. 461–502.

— (2004). “Hypercomputation: philosophical issues”. In: *Theoretical Computer Science* 317, pp. 251–267.

— (2020). “The Church-Turing thesis”. In: *The Stanford Encyclopedia of Philosophy*. Ed. by E. N. Zalta. Summer 2020. URL: https://plato.stanford.edu/archives/sum2020/entries/church-turing/.

Copeland, B. J. and D. Proudfoot (1999). “Alan Turing’s forgotten ideas in computer science”. In: *Scientific American* 280, pp. 77–81.

Copeland, B. J. and R. Sylvan (1999). “Beyond the universal Turing machine”. In: *Australasian Journal of Philosophy* 77, pp. 46–66.

Crane, T. (2003). *The Mechanical Mind*. 2nd ed. London: Routledge.

Cutland, N. (1980). *An Introduction to Recursive Function Theory*. Cambridge: Cambridge University Press.

Davis, M. (2004). "The myth of hypercomputation". In: *Alan Turing: Life and Legacy of a Great Thinker*. Ed. by C. Teuscher. Berlin: Springer, pp. 195–211.

Dean, W. (2016). "Algorithms and the mathematical foundations of computer science". In: *Gödel's Disjunction: The Scope and Limits of Mathematical Knowledge*. Ed. by L. Horsten and P. Welch. Oxford: Oxford University Press, pp. 19–66.

Dennett, D. C. (1978). *Brainstorms*. Cambridge, MA: MIT Press.

Deutsch, D. (1985). "Quantum Theory, the Church–Turing Principle and the Universal Quantum Computer". In: *Proceedings of the Royal Society, Series A* 400, pp. 97–117.

Deutsch, D., A. Ekert and R. Lupacchini (2000). "Machines, Logic and Quantum Physics". In: *Bulletin of Symbolic Logic* 3, pp. 265–283.

Deutsch, D. and R. Jozsa (1992). "Rapid solution of problems by quantum computation". In: *Proceedings of the Royal Society, Series A* 439, pp. 553–558.

Ekert, A. and R. Jozsa (1998). "Quantum algorithms: entanglement-enhanced information processing". In: *Philosophical Transactions of the Royal Society of London, Series A* 356, pp. 1769–1782.

Etesi, G. and I. Németi (2002). "Non-Turing computations via Malament–Hogarth space-times". In: *International Journal of Theoretical Physics* 41, pp. 341–370.

Feynman, R. P. (1982). "Simulating physics with computers". In: *International Journal of Theoretical Physics* 21, pp. 467–488.

Fodor, J. A. (1968). "The appeal to tacit knowledge in psychological explanation". In: *The Journal of Philosophy* 65, pp. 627–640.

Folina, J. (1998). "Church's Thesis: Prelude to a Proof". In: *Philosophia Mathematica* 6, pp. 302–323.

Gandy, R. O. (1980). "Church's thesis and principles of mechanisms". In: *The Kleene Symposium*. Ed. by J. Barwise, H. J. Keisler and K. Kunen. Amsterdam: North Holland, pp. 123–145.

— (1988). "The confluence of ideas in 1936". In: *The Universal Turing Machine: A Half-Century Survey*. Ed. by R. Herken. Oxford: Oxford University Press, pp. 55–111.

Gottesman, D. and I. L. Chuang (1999). "Quantum teleportation is a universal computational primitive". In: *Nature* 402, pp. 390–393.

Gurevich, Y. (1999). “The sequential ASM thesis”. In: *Bulletin of European Association for Theoretical Computer Science* 67, pp. 93–124.

— (2000). “Sequential Abstract State Machines capture sequential algorithms”. In: *ACM Transactions on Computational Logic* 1, pp. 77–111.

— (2011). “What is an algorithm?” In: *SOFSEM 2012: Theory and Practice of Computer Science. Lecture Notes in Computer Science, vol 7147*. Ed. by M. Bieliková, Friedrich G., Gottlob G., Katzenbeisser S. and Turán G. Berlin: Springer, pp. 31–42.

Hosten, O., M. T. Rakher, J. T. Barreiro, N. A. Peters and P. G. Kwiat (2006). “Counterfactual quantum computation through quantum interrogation”. In: *Nature* 949-952.

Johnson-Laird, P. N. (1983). *Mental Models*. Cambridge: Cambridge University Press.

Knuth, D. E. (1972). “Ancient Babylonian algorithms”. In: *Communications of the ACM* 15, pp. 671–677.

— (1981). “Algorithms in modern mathematics and computer science”. In: *Algorithms in Modern Mathematics and Computer Science*. Ed. by A. P. Ershov and D. E. Knuth. Springer-Verlag, pp. 82–99.

— (1997). *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. 3rd ed. Upper Saddle River, NJ: Addison-Wesley.

Lipton, R. J. (1995). “DNA solution of hard computational problems”. In: *Science* 268, pp. 542–545.

Lycan, W. G. (1981). “Form, Function, and Feel”. In: *The Journal of Philosophy* 78, pp. 24–50.

Matuschak, A. and M. A. Nielsen (2019). *Quantum Computing for the Very Curious*. San Francisco, CA. URL: https://quantum.country/qcvc.

Mendelson, E. (1963). “On some recent criticism of Church’s Thesis”. In: *Notre Dame Journal of Formal Logic* 4, pp. 201–205.

Moschovakis, Y. N. (2001). “What Is an algorithm?” In: *Mathematics Unlimited – 2001 and Beyond*. Ed. by B. Engquist and W. Schmid. Berlin: Springer.

Németi, I. and G. Dávid (2006). “Relativistic computers and the Turing barrier”. In: *Applied Mathematics and Computation* 178, pp. 118–142.

Nielsen, M. A. and I. L. Chuang (2010). *Quantum Computation and Quantum Information*. 10th Anniversary. Cambridge: Cambridge University Press.

Oppenheim, P. and H. Putnam (1958). “Unity of science as a working hypothesis”. In: *Concepts, theories, and the mind–body problem*. Ed. by H. Feigl, M. Scriven and G. Maxwell. Minnesota studies in the philosophy of science, Volume 2. Minneapolis, MN: University of Minnesota Press, pp. 3–36.

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

Piccinini, G. (2011). “The physical Church–Turing Thesis: Modest or Bold?” In: *The British Journal for the Philosophy of Science* 62, pp. 733–769.

Putnam, H. (1975). “Philosophy and our mental life”. In: *Mind, Language and Reality, Philosophical Papers, Volume 2*. Cambridge: Cambridge University Press, pp. 291–303.

Rogers, H. (1967). *Theory of Recursive Functions and Effective Computability*. New York, NY: McGraw-Hill.

Searle, J. R. (1980). “Minds, brains, and programs”. In: *Behavioral and Brain Sciences* 3, pp. 417–424.

— (1992). *The Rediscovery of the Mind*. Cambridge, MA: MIT Press.

Shagrir, O. (1998). “Multiple realization, computation and the taxonomy of psychological states”. In: *Synthese* 114, pp. 445–461.

— (2002). “Effective computation by humans and machines”. In: *Minds and Machines* 12, pp. 221–240.

— (2016). “Advertisement for the philosophy of the computational sciences”. In: *The Oxford Handbook of Philosophy of Science*. Ed. by P. Humphreys. draft of chapter for OUP handbook. Oxford: Oxford University Press, pp. 15–42.

Shagrir, O. and I. Pitowsky (2003). “Physical hypercomputation and the Church-Turing thesis”. In: *Minds and Machines* 13, pp. 87–101.

Shapiro, S. (2006). “Computability, proof, and open-texture”. In: *Church’s Thesis After 70 Years*. Ed. by A. Olszewski, J. Woleński and R Janusz. Heusenstamm: Ontos Verlag, pp. 420–455.

Shor, P. W. (1999). “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer”. In: *SIAM Review* 41, pp. 303–332.

Sieg, W. (2002). “Calculation by man and machine: Conceptual analysis”. In: *Reflections on the Foundations of Mathematics (Essays in Honor of Solomon Feferman)*. Ed. by W. Sieg, R. Sommer and C. Talcot. Volume 15 of Lectures Notes in Logic, Association of Symbolic Logic, pp. 387–406.

Smith, P. (2013). *An Introduction to Gödel's Theorems*. 2nd ed. Cambridge: Cambridge University Press.

Soare, R. (1999). "The history and concept of computability". In: *Handbook of Computability Theory*. Ed. by E. R. Griffor. New York, NY: Elsevier, pp. 3–36.

Sprevak, M. (2007). "Chinese rooms and program portability". In: *The British Journal for the Philosophy of Science* 58, pp. 755–776.

Syropoulos, A. (2008). *Hypercomputation: Computing Beyond the Church–Turing Barrier*. New York, NY: Springer.

Tanaka, G., T. Yamane, J. B. Héroux, R. Nakane, Kanazawa N., S. Takeda, H. Numata, D. Nakano and A. Hirose (2019). "Recent advances in physical reservoir computing: A review". In: *Neural Networks* 115, pp. 100–123.

Turing, A. M. (1936). "On computable numbers, with an application to the *Entscheidungsproblem*". In: *Proceeding of the London Mathematical Society, series 2* 42, pp. 230–265.

— (1939). "Systems of Logic Based on Ordinals". In: *Proceedings of the London Mathematical Society, series 2* 45, pp. 161–228.

— (1992). "Proposals for Development in the Mathematics Division of an Automatic Computing Engine (ACE). Report to the Executive Committee of the National Physics Laboratory". In: *Collected Works of A. M. Turing: Mechanical Intelligence*. Ed. by D. C. Ince. Amsterdam: Elsevier, pp. 1–86.

— (2004/1954). "Solvable and unsolvable problems". In: *The Essential Turing*. Ed. by B. J. Copeland. Originally published in *Science News*, 31 (1954), 7–23. Oxford: Oxford University Press, pp. 582–595.

Wittgenstein, L. (1980/1947). *Remarks on the Philosophy of Psychology, Volume 1*. Ed. by G. E. M. Anscombe, H. Nyman and G. H. von Wright. Oxford: Blackwell.

Wu, K., J. Garcıa de Abajo, C. Soci, P. Ping Shum and N. I. Zheludev (2014). "An optical fiber network oracle for NP-complete problems". In: *Light: Science & Applications* 3, e147. DOI: 10.1038/lsa.2014.28.