MIT Press • Open Encyclopedia of Cognitive Science

Computational Complexity

Christopher Mole[1]

[1]University of British Columbia

MIT Press

Published on: Feb 06, 2025

DOI: https://doi.org/10.21428/e2759450.cd81bff2

Designing effective computational systems is often a matter of finding ways in which simple logical operations can be combined to perform more complex tasks. Computer scientists therefore gauge the *complexity* of tasks by asking how many such operations would be needed to perform them. They are especially concerned with cases in which the number of these operations is so large that the task cannot feasibly be performed by computational means. This happens whenever a task that takes n bits of information to specify requires X^n operations to perform. Unless n is very small, the magnitude of the computational resources that would be needed to perform such tasks exceeds anything that could plausibly be available within a brain, or that could have been available over the course of intelligence's evolution. When intelligent creatures succeed in dealing with tasks of this sort, their success therefore cannot be explained simply by the postulation of a computation. We must instead adopt an explanatory strategy that takes into account not only the things that are intelligently done but also the bounds within which this intelligence is accomplished. The question of which tasks exhibit such complexity is among the most fundamental open questions in theoretical computer science.

History

Foundations

In his paper, "Computing Machinery and Intelligence," Alan Turing (1950) laid the foundations for the current phase of our enquiries into the question of whether machines can think. He did so by proposing a test to gauge the presence of thinking [see The Turing Test] and by giving a definition of *machine* that is precise enough to allow for formal enquiry into the limits of what machines can do.

During this phase of his career, Turing was particularly interested in showing that there are some things that no machines could ever possibly do, irrespective of the computational architecture that those machines employ. To establish this, he first described a very simple machine and then showed that, if the working memory that is available to this machine was increased or if each step of the machine's processing were allowed to use increasingly sophisticated logical or mathematical operations, then, although this machine might operate more smoothly, the total set of functions that it could in principle compute would not have changed. By showing that it was impossible for this simple machine to compute certain functions, Turing could thereby show that it would be equally impossible for those functions to be computed by systems that had larger and more sophisticated resources available to them. And so, by finding examples of such functions, he was able to show that it is in the nature of some functions to be uncomputable by any machine, irrespective of that machine's architecture.

In the decade or so following Turing's paper, it became clear that this phenomenon is a more general one. Alan Cobham (1965) argued that just as absolute uncomputability is an intrinsic property of certain functions, so too is the relative difficulty of computing those functions that fall within the realm of the computable. Some functions are impossible to compute. Others are easy to compute, since the quantity of the computational resources that is needed to calculate their output grows only arithmetically, in

proportion to the size of the input that is being considered. Yet others are intrinsically harder to compute, and the quantity of computational resources that must be used to calculate their output grows exponentially as the size of the input increases. If the rate of this exponential growth is bounded so that the exponent remains small, then the computation of these functions may still be manageable (by using a sufficiently fast computational system). But in some cases, there seems to be no such bound, so these are cases in which the demand for computational resources becomes unmanageably large.

To find the prime factors of any number that is n digits long, for example, the only known algorithms need to go through X^n computational steps, but—since the number of milliseconds since the beginning of the universe is something on the order of 4.5×10^{20}—the execution of these algorithms would require more time than could feasibly be made available, unless the number whose prime factors are to be found was rather small, even if the computational system that we were using were one in which each individual computational step could be completed very rapidly.

It is believed that this need for an unmanageably large quantity of computational resources is a consequence of the intrinsic complexity of prime factorization so that prime factorization belongs to the set of functions that are too complex to compute by any efficient means, except by using a machine that can work at speeds that would seem to be unfeasible to implement within the laws of classical physics. There is, however, no proof of this. This reflects the fact that little is known about the set of functions that would be computable only if exponentially large amounts of time were available for their computation. There have been many failed attempts to prove things about this set, and there has been much speculation about it. Kurt Gödel (2003) speculated, in a 1956 letter to John von Neumann, that only a manageable quantity of computational resources might be required for the computation of a function that determines the truth or falsity of a broad range of mathematical conjectures (see Urquhart, 2010). If this were correct, there would be as-yet-undiscovered computational means by which a great many mathematical questions could be settled.

Polynomial vs. nondeterministic polynomial

Research into the issue that Gödel was raising entered a new phase at the beginning of the 1970s, when Stephen Cook (1971) and Leonid Levin (1973) independently investigated the complexity of problems for which the correctness of any proposed solution can be checked easily, whether or not there is any tractable method by which that solution could have been found. For one example of such a problem (suggested by Stewart, 2008), consider the task of finding the solution to any given jigsaw puzzle. Finding the solution to some puzzles might be easy if the pieces are decorated with an obvious pattern, but in other cases—if the pattern is absent or misleading—it would be hard and might require a painstaking search through all of those pieces, in order to find their correct arrangement. For large jigsaws, the time required for such a search might become unfeasible. But in either case—whether the discovery of a proposed solution is easy or hard—the correctness of any solution that has been proposed can easily be checked, just by looking at it to see whether the pieces have been correctly assembled.

Cook and Levin discovered that some of the problems that exhibit such easy checkability have the property that every other easily checkable problem can be expressed as versions of them. If there were a tractable method that could, for example, be used to find whether there is a route by which to make exactly one visit to every station on any given subway network, then that method could be adapted to show, for any possible map of the world, whether there is a way to color that map using only three colors so that no territories with a common border share a color. The opposite also holds: A method for checking the three-colorability of maps could be used to check the visitability of subway networks. And if there were a tractable method for either one of these problems, then it could be also applied to all of the other problems for which proposed solutions can easily be checked. The problems that have this property are said to be *NP-hard*.

The terminology that theoretical computer scientists use to describe the problems that exhibit such properties can be rather opaque. Here, unlike in the parts of cognitive science that take their terminology from linguistics, NP stands for *nondeterministic polynomial*. To understand what this means, consider first what computer scientists mean when they say that some function is "in P" (with "P" here standing for "polynomial"). By "being in P," they mean that two conditions are jointly satisfied. Firstly, they mean that there exists some algorithm that is guaranteed to arrive at the output of that function when given any one of its possible inputs. Secondly, they mean that that algorithm is guaranteed to complete this operation in a number of steps that is never greater than something on the order of N^k, where N is a measure of the size of the particular input that has been given and k is some integer. If this k is sufficiently small then for a function to be in P is for there to be a not-too-onerous route from inputs to outputs. (But this need for sufficient smallness is not built into P's official definition.)

The class NP is best understood by considering the way in which being in NP contrasts with being in P. As compared to saying that a function is in P, saying that it is in NP makes what seems, on the face of it, to be a much weaker claim: It is to say, not that there is a not-too-onerous route from the function's inputs to its outputs, but that there exists some other piece of information that one might have been given (perhaps merely by chance—hence, nondeterministic) such that, if one had somehow been given this other piece of information, there would have been a not-too-onerous route by which one could have used it to establish with certainty what this function's output should be for the input in question.

Although a great many different functions would seem to be in NP, the surprising discovery of theoretical computer science, in the period following Cook's and Levin's pioneering work, is that many of these functions turn out to be notational variants of one another, in the sense that a method for solving any one of them could be used to solve each of the others. So, for example, the checkable questions that can be asked about subway systems can be translated into checkable questions about the colorability of maps, the journeys taken by traveling salesmen, games of minesweeper, and so on (and all of these translations can also be done the other way around). This, finally, enables us to understand what computer scientists mean when they say that a function is NP-hard: A function is NP-hard if and only if, by the application of some such translation scheme, an algorithm for solving it could be applied to every problem in NP.

In the decades following Cook's and Levin's work several problems were shown to have this property of NP-hardness, so that any computational method for solving them would need to be a method that could solve absolutely every problem for which proposed solutions are easily checkable. And many of the problems that have this property are themselves members of NP (in which case they are said to be *NP-complete*). Methods that were guaranteed to solve any one of these problems would, if they existed, be very powerful indeed.

The belief that no method could ever really be so powerful is one of the basic motivations for thinking that computational methods that are guaranteed to solve every instance of these problems do not actually exist. But the underlying point remains unproven, and the question of whether or not such methods really do exist is widely reckoned to be one of the most important open questions in mathematics (Fortnow, 2009). If it were to turn out that computational methods for solving these problems do indeed exist, then those methods would be of the first importance for cognitive science, since they might have the potential to provide the basis for a computational model of general intelligence. Alternatively—on the more widely held supposition that such methods do not exist—the explanation of our human capacity to deal intelligently with these problems must proceed in a more piecemeal fashion, and it must take account of that capacity's limited applicability.

Among the several problems that are known to be NP-hard, there are some basic problems of Boolean logic (Hunt & Stearns, 1990) and of probabilistic inference (Cooper, 1990). There are also a great many problems from domains in which humans are able to cope, competently but fallibly (Hearn & Demaine, 2009). We therefore have a reason for thinking that, whatever computational explanation might be given for our capacity for such coping, there must be some contexts in which that capacity would break down. At the end of the 20th century, such considerations led several cognitive scientists to the view that, if we hope to explain intelligence computationally, we must regard the rationality of that intelligence as bounded (see Simon, 1990) or as achieved through the use of heuristics that are reliable only in a limited range of circumstances (Gigerenzer, 1991) and that are accurate only within some tolerable margin of error (Martignon & Schmitt, 1999).

Core concepts

Foundational assumptions of theoretical computer science

Like other branches of theoretical computer science, the discussion of computational complexity is conducted almost exclusively in the language of *Turing Machines*. To get from theorems that are framed in these terms to claims about the cognitive processes that might be taking place in a brain, we depend on inferences in which some version of the *Church–Turing thesis* serves as a premise. This thesis says that the information processing taking place in any physically possible system can be modeled to an arbitrary degree of precision by a Turing Machine. It is a consequence of Turing's proof that his machines are capable of computing any general recursive function, when that is combined with the assumption that

the only explicable behaviors of physically possible systems are those that conform to mathematically well-behaved laws and that thereby implement such functions.

Because the Church–Turing thesis applies to any physically possible system, it provides a warrant for inferences from claims about the functions that could not possibly be computable by a Turing Machine to conclusions about the capacities that could not be explicably attributed to a brain. It does this without requiring us to make any further assumptions about the brain and the Turing Machine sharing a computational architecture.

The theory of computational complexity is particularly concerned with tasks that become unfeasible to perform on a Turing Machine, not because they are absolutely impossible to compute but only because such machines would need to go through a number of steps that is exponentially larger than the task's input or output, without there being any manageable bound on the size of this exponent. To infer from this that such tasks would be similarly unfeasible for the brain (or for any other physical system at a human scale), one needs to make the further assumption that no such system could have an exponentially large speed advantage over a Turing Machine. This assumption is sometimes known as the *Cobham–Edmonds thesis* (and sometimes just as *Cobham's thesis*). Like the Church–Turing thesis, it is not itself a provable theorem. Instead, it is a contingent claim that licenses the drawing of inferences from such theorems to conclusions about the limitations of what would be physically explicable.

Although the Cobham–Edmonds thesis is consonant with our best physical theories, the physical principles that support it are not so fundamental as those that support the Church–Turing thesis: There may perhaps be counterexamples to it, but the only likely source for these is in quantum mechanics (Shor, 1994). The assumption that the brain can enjoy no more than a polynomial speed advantage over a Turing Machine is therefore a safe one for cognitive science, at least insofar as the brain is too large and too warm for quantum mechanical effects to be relevant to the operation of any system with which it shares information. Theorists who think that quantum mechanical effects in the brain are relevant to the explanation of cognition may think that this assumption should be rejected (see, e.g., Hagan et al., 2002). They might therefore be willing to credit the brain with accomplishments that would be too complex for any classical computer to complete in a feasible amount of time, but even these theorists should not ignore the results of computational complexity theory entirely. Results that are germane to their understanding of cognition might still be found in the (as-yet-nascent) theory of quantum complexity.

Worst-case, fixed-parameter, and average-case complexity classes

Complexity theory's central concept is that of a *complexity class*, which is a set of functions such that, for each of the functions in this set, there is some well-defined criterion that is satisfied by the quantity of the computational resources that would be required to get from the function's input to its output.

The definition for one of the most basic of these classes, known as P, was discussed above: P contains all and only those functions that can be computed in a system that uses a number of computational steps

that is no larger than X^k, where X measures the magnitude of the function's input and k is some natural number. Another related class contains all and only those functions that can be computed in a system where there is a similar bound, not on the number of computational steps that would need to be taken but on the space that would need to be available in the computer's memory. Just as the first of these classes is known as P, the second is known as *PSPACE*. A great many other such classes have been defined. The online "Complexity Zoo" gives the definitions and names of 547 of them.

To understand what is and is not implied by the claim that a function is a member of some hard-to-compute complexity class, it is important to understand the way in which a function's requirement for resources is defined. Even the hardest functions to compute can be made to require only a small amount of work for some circumscribed subset of their instances, just by finding their value for these particular instances in advance and providing one's computational system with a table in which those particular values can be looked up: Computers are always good at looking up previously discovered and previously tested answers, even when they would be bad at freshly calculating those answers for themselves. The magnitude of the resources that are required to find the value of a function in some easy cases therefore tells us nothing very much about that function's intrinsic complexity. For this reason, the complexity of a function is instead typically measured by the magnitude of the resources that are required to compute the value of the function in the worst case.

Because the frequency with which these worst-case scenarios will be encountered in the real world is often unknown, this tendency to focus on complexity classes that are defined by the task-demands of the worst case introduces a limitation on complexity theory's implications for cognitive science. The difficulty here is that we rarely know how lucky one would need to be in order to avoid the difficult cases, not by adopting some clever policy for avoiding them but just by never happening to come across them. As a result of that, we rarely know whether the brain might be able to rely on relatively simple computational methods, even when it is dealing with hard-to-compute problems, just because the organism to whom that brain belongs never needs to deal with the hard cases.

Worst-case analysis is not always useful for cognitive scientists. Consider, for example, some theorems that have been proven about the complexity of the game of Tetris: It has been shown that certain questions about Tetris belong to a complexity class that is believed to be hard to compute in the worst case, even within any known margin of error (Breukelaar et al., 2004). Any computational system for estimating how many lines can be completed using any given sequence of Tetris blocks would therefore be prone to making arbitrarily large errors, if it encountered the worst-case scenario. The proof of this proceeds by showing that NP-hard logic problems can be translated into Tetris block-sequences (thereby showing that an optimal Tetris playing system would need to be able to solve those problems). But this proof tells us nothing about how hard it might be to use the sorts of computational resources that are available within a human brain to do a close-to-optimal job of making lines from the randomly generated sequences of Tetris blocks that are encountered by actual players of the game. The worst-case scenarios

that are used in this proof are designed to be hard, not to be representative of any normally occurring case.

Membership of a hard-to-compute complexity class can reveal something that is more relevant to the concerns of the cognitive scientist when the outputs of the functions in that class are hard to compute, not only in the absolutely worst case but also in the worst of the cases that are encountered when certain parameters are fixed within naturally occurring bounds. It may, for example, be unfeasible to design a computational system that can check the logical consistency of any set of sentences, but it would be easy to design a system that can check the consistency of any set that is small enough to be held in human working memory. It has therefore been suggested that our theories of cognition should be constrained, not by the absolute complexity of the functions that a cognitive system would need to compute, but only by the complexity of those functions in cases where certain parameters are fixed (van Rooij, 2008). This is a much weaker constraint. Many functions that would be hard to compute with full generality become easy to compute if we consider only those instances that occur within parameters that are fixed to psychologically plausible values.

Membership of a hard-to-compute complexity class also promises to reveal something that is relevant to the concerns of the cognitive scientist when the values of the functions in that class are hard to compute, not only in the worst case but also in the average case. Theorems about average-case complexity are, however, much harder to prove. Unlike facts about worst-case complexity (or worst-case complexity within fixed parameters), facts about a function's average-case complexity cannot be intrinsic to that function: They must instead be relativized to an expected probability distribution of the cases to which that function is going to be applied. Cognitive scientists who are hoping to use average-case complexity to cast light on the actual limits of our cognitive processes therefore need to make assumptions about those distributions.

Of the natural assumptions that might be made here, one of the few that has been made precise enough for mathematical purposes is the assumption that nothing in our environment does unfeasible amounts of computational work in order to ensure that we are presented with scenarios that are likely to be hard or easy for us. On the basis of that assumption, it is possible to show that there is sometimes no feasible way in which we could improve our chances of avoiding the worst-case scenario (Impagliazzo & Levin, 1990). But, for the kinds of problems that need to be intelligently managed in the course of a normal human life, we still know rather little about how good or bad those chances are (Mole, 2016).

Questions, controversies, and new developments

The most direct way in which complexity theory could reveal something that would be of fundamental importance for cognitive science would be if it could identify some naturally arising problems that satisfy three criteria: (1) The computation of solutions would be hard for a significant proportion of the cases that we are likely to encounter in the course of a normal human life; (2) the computation of those solutions would remain hard, even if we could tolerate solutions that were approximate to within some

human-like margin for error; and (3) the finding of those solutions would not be speeded up if we allowed them to be found in the chancy way that seems to be typical of human thinking. An explanation of our ability to cope with problems that meet all three of these criteria would need to depart from cognitive science's established explanatory paradigms, but results concerning the average-case complexity of such naturally arising problems remain elusive.

One area in which considerable progress has been made is in understanding the complexity-theoretic consequences of allowing for chanciness. One might naively have supposed that, since Turing Machines are deterministic, the theory of computational complexity ceases to be applicable to any system that makes use of randomness (as the brain might). But this would be a mistake. It can be shown, conditional on certain plausible assumptions, that genuine randomness has no real advantages over the kinds of pseudorandomness that Turing Machines are capable of generating. The lessons that come from the study of Turing Machines are therefore applicable even to systems that make use of randomness (Impagliazzo & Wigderson, 1997). There are many contexts in which these systems prove to be surprisingly powerful. Research into randomness-employing methods has revealed some deep connections between tractability by such methods and the hardness of approximation. Some of the most active areas of research in complexity theory build on these discoveries.

Much of that research is concerned with identifying problems that display average-case hardness and that are hard to even approximate, but only a small part of this research is oriented toward questions concerning the computational explanation of cognition (see, e.g., Bossaerts & Murawski, 2017). Much of it is instead concerned with the discovery of functions that can be used to generate codes that lie far beyond the range of what could be deciphered by a human, even if that human had enormous computational resources at their disposal, and that can therefore be used to create secure systems of encryption, for use in various cybersecurity applications.

Broader connections

In addition to complexity theory's promise as a source of future insights into the limitations of our intelligence, complexity theoretic considerations might also cast light on some other features of the mind. Theoretical computer science is a rich source of examples of deterministic processes, the outcome of which cannot be predicted in advance (except by a perfect replication of that very process). The existence of such processes removes one of the reasons why deterministic laws of nature might have been thought to be incompatible with the existence of free will. Some researchers in complexity theory have therefore advanced the philosophically ambitious suggestion that the sources of unpredictability that are studied by computational complexity theory might be used to explain how systems obeying deterministic laws can manifest something like free will (see Aaronson, 2013, Chapter 19).

Connections with other topics in cognitive science have been drawn in the branch of theoretical computer science that is concerned with the study of *ecorithms*, a term coined by Leslie Valiant to refer to algorithms that could enable naturally occurring systems to efficiently extract information from their

environment and to encode that information in a usable form. Such ecorithms have been used to model the way in which an individual creature comes to possess information about its environment over the course of its history of inductively learning from examples. They have been used to model the acquisition of concepts from relatively small numbers of positive and negative exemplars and to model some aspects of the acquisition of language (see the papers in Bertolo, 2001). At a higher level of abstraction, this approach has also been used to model the way in which the process of evolution can incrementally create an information-rich lineage of DNA (Valiant, 2009).

Considerations of computational complexity feature prominently in the study of ecorithms, since it is the need to avoid unfeasibly complex tasks that gives this study its characteristic emphasis on the fact that the algorithms unpinning these ecological phenomena are effective with a high probability, rather than with complete certainty, and produce representations that have tolerable margins of error but may not have complete fidelity to the underlying facts. These two emphases are reflected in the fact that ecorithms are said to use techniques of *Probably Approximately Correct (PAC) learning* (Valiant, 2013). It has also been suggested that the theory of ecorithms might be illuminating at a more metatheoretical level: Concepts that were acquired through PAC learning processes would have a somewhat indeterminate semantics, and the suggestion has been made that some philosophical questions about the mind are intractable owing to such indeterminacy (Valiant, 2013, Chapter 8).

The parts of cognitive science that make use of game theory also have the potential to be illuminated by research into computational complexity. Game theoretic considerations lead us to expect that certain strategies for negotiating multiagent interactions will tend to be found in any stable population. In particular, they lead us to expect that the behavior of stable populations will exemplify some combination of strategies such that none of the agents participating in this interaction could improve their outcome by deviating from their current strategy. These equilibrium states can, however, be difficult to find. The field of *Algorithmic Game Theory* uses the resource of computational complexity theory to probe the sources of those difficulties (see, e.g., Daskalakiset al., 2009) and to examine the ways in which physically explicable systems might address them over human or evolutionary time scales (Papadimitriou, 2014).

Acknowledgments

Thanks to the editors of the OECS for feedback on earlier drafts.

Further reading

- Aaronson, S. (2013). *Quantum computing since Democritus*. Cambridge University Press. https://doi.org/10.1017/CBO9780511979309
- Arora, S., & Barak, B. (2009). *Computational complexity: A modern approach.* Cambridge University Press. https://doi.org/10.1017/CBO9780511804090
- Goldreich, O. (2012). Invitation to complexity theory. *XRDS Crossroads: The ACM Magazine for Students, 18*(3), 18–22. https://doi.org/10.1145/2090276.2090285

- Valiant, L. G. (2013). *Probably approximately correct*. Basic Books.

References

- Aaronson, S. (2013). *Quantum computing since Democritus*. Cambridge University Press. https://doi.org/10.1017/CBO9780511979309 ↩
- Bertolo, S. (2001). *Language acquisition and learnability*. Cambridge University Press. ↩
- Bossaerts, P., & Murawski, C. (2017). Computational complexity and human decision-making. Trends in Cognitive Sciences, *21*(12), 917–929. https://doi.org/10.1016/j.tics.2017.09.005 ↩
- Breukelaar, Ron, Erik D. Demaine, Susan Hohenberger, Hendrik Jan Hoogeboom, Walter A. Kosters, and David Liben-Nowell (2004). "Tetris is hard, even to approximate." *International Journal of Computational Geometry & Applications, 14* (1-2): 41-68. https://doi.org/10.1142/S0218195904001354 ↩
- Cobham, A. (1965). The intrinsic computational difficulty of functions. In Y. Bar-Hillel (Ed.), *Logic, methodology, and philosophy of science* (pp. 24–30). North Holland Publishing. ↩
- Cook, S. A. (1971). The complexity of theorem proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing* (pp. 151–158). Association for Computing Machinery. https://doi.org/10.1145/800157.805047 ↩
- Cooper, G. F. (1990). The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, *42*(2–3), 393–405. https://doi.org/10.1016/0004-3702(90)90060-D ↩
- Daskalakis, C., Goldberg, P. W., & Papadimitriou, C. H. (2009). The complexity of computing a Nash equilibrium. *Communications of the ACM, 52*(2), 89–97. https://doi.org/10.1145/1461928.1461951 ↩
- Fortnow, L. (2009). The status of the P versus NP problem. *Communications of the ACM*, *52*(9), 78–86. https://doi.org/10.1145/1562164.1562186 ↩

- Gigerenzer, G. (1991). How to make cognitive illusions disappear: Beyond "heuristics and biases." *European Review of Social Psychology*, *2*(1), 83–115. https://doi.org/10.1080/14792779143000033

↩

- Gödel, K. (2003). *Kurt Gödel Collected works: Volume V: Correspondence H–Z* (S. Feferman, J. W. Dawson Jr., W. Goldfarb, C. Parsons, and W. Sieg, Eds.). Oxford University Press.

↩

- Hagan, S., Hameroff, S. R., & Tuszyński, J. A. (2002). Quantum computation in brain microtubules: Decoherence and biological feasibility. *Physical Review E*, *65*(6), 061901. https://doi.org/10.1103/PhysRevE.65.061901

↩

- Hearn, R. A., & Demaine, E. D. (2009). *Games, puzzles, and computation*. CRC Press. https://doi.org/10.1201/b10581

↩

- Hunt, H. B., III, & Stearns, R. E. (1990). The complexity of very simple Boolean formulas with applications. *SIAM Journal on Computing*, *19*(1), 44–70. https://doi.org/10.1137/0219003

↩

- Impagliazzo, R., & Levin L. A. (1990). No better ways to generate hard NP instances than picking uniformly at random. In *Proceedings 31st Annual Symposium on Foundations of Computer Science* (pp. 812–821). IEEE. https://doi.org/10.1109/FSCS.1990.89604

↩

- Impagliazzo, R., & Wigderson, A. (1997). *P* = *BPP* if *E* requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing* (pp. 220–229). Association for Computing Machinery. https://doi.org/10.1145/258533.258590

↩

- Levin, L. A. (1973). Universal search problems. *Problems of Information Transmission*, *9*(3), 265–266.

↩

- Martignon, L., & Schmitt, M. (1999). Simplicity and robustness of fast and frugal heuristics. *Minds and Machines*, *9*, 565–593. https://doi.org/10.1023/A:1008313020307

↩

- Mole, C. (2016). *The unexplained intellect: Complexity, time, and the metaphysics of embodied thought*. Routledge.

↩

- Papadimitriou, C. (2014). Algorithms, complexity, and the sciences. *Proceedings of the National Academy of Sciences*, *111*(45), 15881–15887. https://doi.org/10.1073/pnas.1416954111

  ↩
- Shor, P. W. (1994). Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings of the 35th annual symposium on foundations of computer science* (pp. 124-134). IEEE.

  ↩
- Simon, H. A. (1990). Invariants of human behavior. *Annual Review of Psychology*, *41*(1), 1–19. https://doi.org/10.1146/annurev.ps.41.020190.000245

  ↩
- Stewart, I. (2008). *Professor Stewart's cabinet of mathematical curiosities*. Basic Books.

  ↩
- Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, *59*(236), 433–469. https://doi.org/10.1093/mind/LIX.236.433

  ↩
- Urquhart, A. (2010). Von Neumann, Gödel and complexity theory. *Bulletin of Symbolic Logic*, *16*(4), 516–530. https://doi.org/10.2178/bsl/1294171130

  ↩
- Valiant, L. G. (2009). Evolvability. *Journal of the ACM*, *56*(1), 3. https://doi.org/10.1145/1462153.1462156

  ↩
- Valiant, L. G. (2013). *Probably approximately correct*. Basic Books.

  ↩
- van Rooij, I. (2008). The tractable cognition thesis. *Cognitive Science*, *32*(6), 939–984.

  ↩