Ronald de Wolf
CWI, Amsterdam, and
ILLC, University of Amsterdam
rdewolf@cwi.nl

Event **Abel Prize 2021**

Avi Wigderson's work and influence

The 2021 Abel Prize was awarded to Lászlo Lovász and Avi Wigderson "For their foundational contributions to theoretical computer science and discrete mathematics, and their leading role in shaping them into central fields of modern mathematics." While Lovász is a mathematician whose work affects computer science, Wigderson is a computer scientist whose work affects mathematics. In this article Ronald de Wolf gives a brief overview of some highlights of Wigderson's wide-ranging work and influence.

Avi Wigderson was born in 1956 in Haifa, Israel, where he studied at the Technion before going to Princeton to obtain his PhD in 1983 under Richard Lipton. After several short-term positions in the US and a long period on the faculty of the Hebrew University in Jerusalem, in 1999 he joined the School of Mathematics of the Institute for Advanced Studies (IAS) in Princeton. There, despite being the only tenured computer scientist, he has created a thriving research environment for theoretical computer science and its interplay with mathematics and other areas. He is famous for his openness to everything interesting, his willingness to collaborate widely, and his ability to mentor young researchers.

Wigderson already received the 1994 Nevanlinna Prize for work on complexity theory, the 2009 Gödel Prize for work on graph products, and the 2019 Knuth Prize for his overall contributions to the foundations of computer science, but this year's well-deserved Abel Prize is arguably an even more prestigious honour.

Proofs, with and without knowledge

Mathematics is all about proofs, and because complexity theorists dislike hard-to-check proofs, much of their field focuses on proofs that are *efficiently verifiable*, i.e., checkable in polynomial time. The quintessential incarnation of this is the complexity class NP, which consists of those decision problems where the 'yes-instances' have a short and efficiently-verifiable certificate. Such problems may not always be efficiently solvable (that's what the famous $\mathrm{P} \neq \mathrm{NP}$ conjecture says), but at least we can efficiently verify a given solution.

One example of such a computational decision problem, at the very core of mathematics, is the set of statements that have a short proof (i.e., a proof with at most polynomially many derivation steps starting from the axioms in some fixed formal system). *Given* a purported proof of some statement, one can efficiently verify that the proof is correct by checking that it starts from the system's axioms, that each step follows the derivation rules of the system, and that the conclusion is the claimed statement. However, deciding whether a given statement *has* such a proof or even *finding* such a proof may of course be much more difficult, keeping mathematicians busy in their daily life in creative ways that are very hard to automate.

The practice of mathematical proofs is often more complex than the simple "give me a proof and I'll verify it" scheme of NP. Proofs are often *interactive*: instead of a static proof that the prover gives to the verifier, in an interactive proof the verifier can ask questions, ask the prover to elucidate some points, et cetera. This involves back and forth communication until the verifier accepts or rejects the statement that the prover is trying to prove. For instance, this is the situation when a mathematician presents a proof to colleagues on a whiteboard or during a seminar with an active audience, or sometimes in the interaction with an anonymous referee when a paper is submitted to a journal. Such interactive proofs have been modelled mathematically in the complexity class IP: here the verifier is required to work efficiently while interacting with a prover who has unbounded computational power. A set L of strings has an interactive proof system if the prover can convince the verifier of each statement in L but of none of the statements outside of L. A fundamental result in complexity theory says that $\mathrm{IP} = \mathrm{PSPACE}$ [19,26,27]: a set L has an interactive proof system iff there is a deterministic algorithm that uses polynomial space (and possibly much more time) for deciding if a given string is in L.

Of course, a proof of a statement, no matter whether interactive or not, typically gives us a lot of knowledge about *why* the statement is true. If $\mathrm{P} \neq \mathrm{NP}$ (as most researchers expect) then presumably it would have taken the verifier a lot of time to *find* that knowledge without the help of the prover. Rather amazingly, there are also proof systems *that give no knowledge* beyond the validity of the proved statement. These are known as *zero-knowledge* proofs.

An illustrative example is the problem of deciding whether two given n-vertex graphs, G_0 and G_1, are isomorphic or not. This problem is clearly in NP: the proof is the isomorphism (easy to verify, though maybe hard to find). On the other hand, it's hard to think of a succinct proof that two graphs are *not* isomorphic, and we do not even know whether such succinct proofs exist (i.e., whether the graph non-isomorphism problem is in NP). However, here is a simple zero-knowledge proof system for checking with high confidence that G_0 and G_1 are *not* isomorphic: the verifier flips a random coin $b \in \{0,1\}$, sends the prover a random permutation of the graph G_b, and asks if it came from G_0 or G_1. In other words, the prover has to guess what b was. If the two graphs are non-isomorphic, then the prover can (in exponential time) determine whether the graph he received came from G_0 or G_1 and tell the verifier what b was. If, on the other hand, G_0 and G_1 were isomorphic then the prover just sees a random graph isomorphic to *both* G_0 and G_1, and his best bet at guessing b is a random coin flip — which will be wrong half the time. Repeating this a few times, with the verifier choosing a new random bit b each time, the prover can correctly guess these bits b every time if G_0 and G_1 are not isomorphic, but he's very likely to guess wrong at least once if G_0 and G_1 are isomorphic.

Can we find zero-knowledge proof systems for other, more difficult problems than non-GI? We do not know an efficient algorithm for non-GI (the best is Babai's fairly recent algorithm [4]) but neither is it believed to be among the hardest problems in NP: it is unlikely to be NP-hard. Rather amazingly, Wigderson and co-authors proved in [7,11] that there is an efficient zero-knowledge proof system *for every problem in NP*. In particular, there is a way for a prover to convince a verifier that a certain statement in a formal system is provable *without revealing any information about that proof beyond the fact that it exists*. What does the latter mean, exactly? It means that the verifier can himself, without help from the prover, simulate the whole conversation if the statement is true. For example, in the earlier protocol for non-isomorphism of two graphs, the verifier already knows in advance what the prover will reply in the non-isomorphic case: the prover will just tell the verifier what b is — which the verifier already knew, since he chose b himself!

These results about zero-knowledge proofs and their further development are among the many ways in which Wigderson has impacted the closely related areas of computational complexity, proof theory, and cryptography.

Communication complexity

The area of *communication complexity* was introduced by Yao [29]. It studies a bare-bones version of distributed computing, where the only resource we care about is the amount of communication, while all local computation is for free. In its most basic two-party setting, Alice and Bob receive inputs x and y, respectively, and want to compute some function $f(x,y)$ that depends on both of their inputs. The function f is known to both players, but x is known only to Alice and y is known only to Bob, so they will have to communicate at least something if they want to compute $f(x,y)$. Of course, Alice could just send x to Bob or Bob could send y to Alice, but these may be long strings, and for many functions much more clever and communication-efficient ways of computing $f(x,y)$ exist. The goal here is to analyze the minimal number of bits of communication needed, on the worst-case input x, y.

The function f corresponds to a matrix M_f, with rows indexed by the possible x's, columns indexed by the possibly y's, and entries $f(x,y)$. Since M_f is a complete description of f, its properties determine the communication complexity of f. Which properties exactly? Many connections have been found between the communication complexity of f and mathematical properties of M_f, for instance combinatorial properties like the minimal number of 'monochromatic rectangles' (submatrices of M_f on which f is constant) needed to partition or to cover M_f, and algebraic properties such as various types of rank of M_f. A rich array of mathematical techniques can thus be deployed to analyze the communication complexity of a given function. Wigderson contributed much to this in a number of papers together with Noam Nisan [15,20,22].

Apart from being a basic model of distributed computation, communication complexity is also one of our main sources of lower bounds in many other models, ranging from area-time tradeoffs in chips to decision trees to data structures. One very surprising connection is due to Karchmer and Wigderson [16], who gave a very clean characterization of the minimal Boolean circuit depth of a given function in terms of the communication complexity of a related problem. In principle this allows us to prove lower bounds on circuit depth via communication complexity, and such circuit lower bounds could eventually lead to separation of complexity classes like P and NP. Unfortunately the resulting communication complexity problems are still quite hard to analyze, and this approach has not yet led to super-logarithmic lower bounds on circuit depth. It has, however, led to the first super-logarithmic lower bounds on circuit depth of *monotone* Boolean circuits for certain graph problems [16,23].

Pseudorandomness and derandomization

Another important strand in Wigderson's research is the role of randomness in computation.

Computers, in Turing's definition, in actual hardware, and in the general public's perception, are supposed to be *deterministic* machines: one computational operation follows the previous one with perfect predictability. If anything makes the computer deviate from that one path, then that's called noise or a 'bug' (sometimes literally). However, since the 1970s *randomized* algorithms have become ubiquitous in theory and in some parts of practice (for instance in cryptography and in Monte Carlo simulations). Randomized algorithms are allowed to let their computational path depend on coin flips or other sources of randomness, and are typically allowed to have a small error probability on each input. Usually that error probability can be reduced very efficiently to something extremely small by repeating the algorithm a few times and taking the majority output among those runs. So for most practical purposes, an efficient randomized algorithm is as good as an efficient deterministic one.

Here is a simple example from communication complexity where randomization really helps: suppose Alice and Bob have n-bit integers x and y, respectively, and they want to compute the 'equality' function, i.e., decide if $x=y$ or not. One can show that deterministic communication protocols need to send n bits across for this. In contrast, if Alice can flip a coin then she can do the following: she chooses a random prime number p of $O(\log n)$ bits, and sends both p and $x \bmod p$ to Bob. That's just $O(\log n)$ bits of communication. Bob receives these, computes $y \bmod p$, and compares it with $x \bmod p$. Clearly, if $x=y$ then the same is true $\bmod p$. But if $x \neq y$ then x and y will be different $\bmod p$ with high probability!

The above example shows that randomness can yield provable exponential savings in terms of communication: allowing the parties to flip coins and to have a tiny error probability reduces the communication complexity of the equality function from n to $O(\log n)$ bits. Randomness also helps to *hide* things from an adversary in cryptographic situations, as we saw for instance for zero-knowledge proofs. In fact, without the ability to use randomness to choose secret private keys, there would be no cryptography.

What about the power of randomness for the core concept of complexity theory: algorithmic runtime? Can randomization lead to an exponential speed-up compared to deterministic algorithms for some computational problems? After a wave of efficient randomized algorithms that we didn't (and in some cases still don't) know how to replace by an efficient deterministic algorithm, it seemed for a while quite plausible that the answer is 'yes'. In complexity-theoretic terms, this would mean that the class BPP of problems efficiently solvable by *randomized* algorithms would be strictly larger than the class P of problems efficiently solvable by *deterministic* algorithms. (Consider the following 'polynomial identity testing' (PIT) problem. Given a low-degree multivariate polynomial $p(x_1,\dots,x_n)$ over some field, represented as an arithmetic circuit — i.e., a circuit acting on n inputs, whose internal nodes are addition and multiplication operations —, decide if p is identically equal to 0 or not. By the Schwartz–Zippel lemma, the value of the polynomial on a uniformly random input will be non-zero with reasonably high probability, unless the polynomial was identically equal to 0. This gives an efficient *randomized* algorithm for PIT: evaluate the arithmetic circuit on a random input, and see if you get value 0. Challenge: find an efficient *deterministic* algorithm ...)

Turning this expectation around, a sequence of so-called *derandomization* papers in the 1990s, first by Nisan and Wigderson [21] and then by Impagliazzo and Wigderson [13,14] showed that P and BPP are actually equal (meaning efficient randomized algorithms can be replaced by efficient deterministic ones) under quite plausible hardness assumptions, such as that there are problems solvable in deterministic time $2^{O(n)}$ that require Boolean circuits of size $2^{\Omega(n)}$. The magic of this approach is that the truth-table of such a hard function can be used to construct a *pseudorandom* generator g that efficiently stretches an $O(\log n)$-bit uniformly random seed s to a polynomial-length 'pseudorandom' string $g(s)$. 'Pseudorandom' here means that an efficient algorithm cannot 'see' the difference between a truly random string and an equally-long pseudorandom string $g(s)$ that is generated from a small uniformly random seed s. Accordingly, the success probability of a randomized algorithm wouldn't change significantly if we fed it a pseudorandom string rather than the truly random string that it expects. But then the strategy to make a given randomized algorithm deterministic is obvious: a deterministic algorithm can, in polynomial time, go over all $2^{O(\log n)}=n^{O(1)}$ seeds s, run the no-longer-randomized algorithm on the string $g(s)$, and see which output value occurs most often among those polynomially-many runs. The upshot here is that, surprisingly and in contrast to the situation in communication complexity, randomness does not seem to confer much additional power in computational complexity.

Graphs, algebra and algorithms

The last broad area of Wigderson's research that I'd like to highlight here is research combining graph theory, algebra and algorithmics. These connections work both ways: results from algebra and graph theory help design algorithms, and much of the work in algorithms and complexity leads to new insights in algebra and graph theory.

For example, Wigderson and co-authors developed efficient algorithms for graph-theoretic problems such as matching [17], whose analysis uses algebra. They also designed algorithms for algebraic problems such as matrix scaling [2,18] and operator scaling [8,10], where the analysis involves graph theory. Often the connection is through algebraic graph theory: the algebraic properties of a graph's adjacency matrix give crucial information about the graph, while conversely every $n \times n$ matrix can be viewed as the adjacency matrix of a weighted n-vertex graph.

Another example of such connections are the many uses of *expander graphs*. These are constant-degree graphs that are like a 'poor man's version' of the complete graph. On the one hand an expander shares many desirable properties with the complete graph, such as having short distances between any two vertices, and rapid mixing of random walks. On the other hand, because the graph has constant degree (degree 3 already suffices!) the number of required edges is only linear in the number of vertices instead of quadratic as in the complete graph. Such graphs have two equivalent definitions: a graph-theoretic one where we require every set of at most half the vertices to 'expand' (i.e., to be connected to many new vertices), and an algebraic one where we require the adjacency matrix's second eigenvalue to be somewhat smaller than the first. The interplay between these graph-theoretic and algebraic perspectives has been very fruitful. Wigderson has been instrumental in developing constructions and applications of such expander graphs, as well as of their cousins such as extractor and disperser graphs, which have their applications in derandomization and even in pure graph theory such as the construction of explicit Ramsey graphs [5]. See his beautiful survey [12] for much more.

The 'zig-zag product' of graphs of Reingold, Vadhan and Wigderson [25] deserves to be mentioned here. This is a way to combine a large graph with a small graph to get an even larger graph that inherits the expansion properties of the small graph. It was instrumental in two subsequent breakthroughs: Reingold's proof that deciding whether two vertices are connected on a given graph can be solved deterministically using only logarithmic space [24], and Dinur's new proof of the PCP theorem via gap-amplification [9]. (The PCP theorem [3] says that proofs for NP-problems can always be written in such a way that a randomized verifier can check them by looking at only a *constant* number of the bits of the proof! This is a deep and very surprising result about proof checking, which also has many applications for showing hardness of approximation problems.)

The zig-zag product has also led to progress in graph theory itself, including explicit constructions of almost-Ramanujan graphs [6] (i.e., nearly optimal expanders).

Together with Scott Aaronson, Wigderson [1] also proved some strong limitations on what algebraic methods can achieve in complexity theory: the algebraic technique of writing functions as low-degree multivariate polynomials, which has been crucial for results like the aforementioned $\mathrm{IP} = \mathrm{PSPACE}$, by itself will not be enough to prove long-hoped-for results like $\mathrm{P} \neq \mathrm{NP}$.

TCS as a lens on other areas

In addition to the strands of research highlighted above, Wigderson has also worked on parallel computing, data structures, quantum computing, pure graph theory, and many other areas. He has a very broad perspective on the theory of computing, with a keen eye for how it can learn from, interact with, and illuminate other parts of science.

The fact that there are many links between TCS and mathematics, some obvious and some very surprising, is by now well-known, in part through Wigderson's many able and Abel contributions. However, TCS also has growing relevance for and impact on other areas such as physics (quantum computing and quantum information theory, threshold phenomena in statistical physics, ...), biology (biocomputing, the view of DNA as an information-carrier, ...), economics (design and analysis of auctions, the complexity of finding Nash equilibria, ...), cognitive science (learning theory, neural networks, ...) and many other areas. A synthesis of this broad perspective is Wigderson's recent book *Mathematics and Computation* [28], which is warmly recommended for anyone interested in theoretical computer science, both for its own sake and as a lens on mathematics and other sciences.

Acknowledgments
Thanks to Robbert Fokkink for some helpful feedback.

References

1 S. Aaronson and A. Wigderson, Algebrization: A new barrier in complexity theory, *ACM Transactions on Computation Theory* 1(1) (2009), 2:1–2:54. Earlier version in STOC'08.

2 Z. Allen-Zhu, Y. Li, R. Oliveira and A. Wigderson, Much faster algorithms for matrix scaling, in *Proceedings of 58th IEEE FOCS*, 2017, pp. 890–901.

3 S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, Proof verification and the hardness of approximation problems, *Journal of the ACM* 45(3) (1998), 501–555. Earlier version in FOCS'92.

4 L. Babai, Graph isomorphism in quasipolynomial time, in *Proceedings of 48th ACM STOC*, 2016, pp. 684–697.

5 B. Barak, A. Rao, R. Shaltiel and A. Wigderson, 2-source dispersers for $n^{o(1)}$ entropy, and Ramsey graphs beating the Frankl–Wilson construction, *Annals of Mathematics* 176 (2012), 1483–1543. Earlier version in STOC'08.

6 A. Ben-Aroya and A. Ta-Shma, A combinatorial construction of almost-Ramanujan graphs using the zig-zag product, *SIAM Journal on Computing* 40(2) (2011), 267–290. Earlier version in STOC'08.

7 M. Ben-Or, S. Goldwasser, J. Kilian and A. Wigderson, Multi-prover interactive proofs: How to remove intractability assumptions, in *Proceedings of 20th ACM STOC*, 1988, pp. 113–131.

8 P. Bürgisser, C. Franks, A. Garg, R. Oliveira, M. Walter and A. Wigderson, Efficient algorithms for tensor scaling, quantum marginals, and moment polytopes, in *Proceedings of 59th IEEE FOCS,* 2018, pp. 883–897.

9 I. Dinur, The PCP theorem by gap amplification, *Journal of the ACM* 54(3) (2007), 12. Earlier version in STOC'06.

10 A. Garg, L. Gurvits, R. Oliveira and A. Wigderson, Operator scaling: theory and applications, *Foundations of Computational Mathematics* 20(2) (2020), 223–290.

11 O. Goldreich, S. Micali and A. Wigderson, Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems, *Journal of the ACM,* 38(3) (1991), 691–729. Earlier version in FOCS'86.

12 S. Hoory, N. Linial and A. Wigderson, Expander graphs and their applications, *Bulletin of the AMS* 43 (2006), 439–561.

13 R. Impagliazzo and A. Wigderson, $\mathrm{P} = \mathrm{BPP}$ if E requires exponential circuits: Derandomizing the XOR lemma, in *Proceedings of 29th ACM STOC*, 1997, pp. 220–229.

14 R. Impagliazzo and A. Wigderson, Randomness vs time: Derandomization under a uniform assumption, *Journal of Computer and System Sciences* 63(4) (2001), 672–688. Earlier version in FOCS'98.

15 M. Karchmer, I. Newman, M. Saks and A. Wigderson, Non-deterministic communication complexity with few witnesses, *Journal of Computer and System Sciences,* 49(2) (1994), 247–257. Earlier version in Structures'92.

16 M. Karchmer and A. Wigderson, Monotone circuits for connectivity require super-logarithmic depth, *SIAM Journal on Discrete Mathematics* 3(2) (1990), 255–265. Earlier version in STOC'88.

17 R. Karp, E. Upfal and A. Wigderson, Constructing a perfect matching is in random NC, *Combinatorica* 6(1) (1986), 35–48. Earlier version in STOC'85.

18 N. Linial, A. Samorodnitsky and A. Wigderson, A deterministic strongly polynomial algorithm for matrix scaling and approximate permanents, *Combinatorica* 20(4) (2000), 545–568. Earlier version in STOC'98.

19 C. Lund, L. Fortnow, H. Karloff and N. Nisan, Algebraic methods for interactive proof systems, *Journal of the ACM* 39(4) (1992), 859–868. Earlier version in FOCS'90.

20 N. Nisan and A. Wigderson, Rounds in communication complexity revisited, *SIAM Journal on Computing* 22(1) (1993), 211–219. Earlier version in STOC'91.

21 N. Nisan and A. Wigderson, Hardness vs. randomness, *Journal of Computer and System Sciences* 49(2) (1994), 561–570. Earlier version in FOCS'88.

22 N. Nisan and A. Wigderson, On rank vs. communication complexity, *Combinatorica* 15(4) (1995), 557–565. Earlier version in FOCS'94.

23 R. Raz and A. Wigderson, Monotone circuits for matching require linear depth, *Journal of the ACM* 39(3) (1992), 736–744. Earlier version in STOC'90.

24 O. Reingold, Undirected connectivity in logspace, *Journal of the ACM* 55(4) (2008), 17:1–17:24. Earlier version in STOC'05.

25 O. Reingold, S. Vadhan and A. Wigderson, Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors, in *Proceedings of 41st IEEE FOCS*, 2000, pp. 3–13.

26 A. Shamir, $\mathrm{IP} = \mathrm{PSPACE}$, *Journal of the ACM* 39(4) (1992), 869–877. Earlier version in FOCS'90.

27 A. Shen, $\mathrm{IP} = \mathrm{PSPACE}$: Simplified proof, *Journal of the ACM* 39(4) (1992), 878–880.

28 A. Wigderson, *Mathematics and Computation A Theory Revolutionizing Technology and Science*, Princeton University Press, 2019.

29 A.C.-C. Yao, Some complexity questions related to distributive computing, in *Proceedings of 11th ACM STOC*, 1979, pp. 209–213.