Monografías de la Real Academia de Ciencias de Zaragoza. **26:** 57–68, (2004).

P versus NP

Elvira Mayordomo

Departamento de Informática e Ingeniería de Sistemas
Universidad de Zaragoza
María de Luna 1, 50018 Zaragoza, SPAIN.
elvira@unizar.es

1 Introduction and history

In 1971 Cook first explicitly formulated the P versus NP conjecture. This is indeed the youngest of the seven Millennium problems, though I strongly believe that it won't be the first, not even the second of them to be solved. I'll go back to this daring statement, but let me start with a brief introduction to the problem and a summary of its history.

The term P, or polynomial time, refers to the class of (decisional) problems that can be efficiently solved by an algorithm. NP, for nondeterministic polynomial time, is the class of (decisional) problems for which "solutions" or certificates are efficiently verifiable. Therefore the question of whether P is equal to NP means whether solving is harder than verifying. Yet in other words, we want to know whether there is always an efficient alternative to brute force search. Formal definitions will be given in the next section.

Historically, Cook [9] and Levin [20] first defined the class NP and proved the existence of complete problems for NP, that is, problems such as SAT for which the question "Is SAT solvable efficiently?" is equivalent to "Is P equal to NP?". Karp [18] demonstrated that many familiar problems were complete for NP.

The root of this work can be traced back to the thirties and Computability or Recursion Theory originated by Turing, Church and Gödel. Computability theory is the immediate precursor of Computational Complexity, that Hartmanis, Lewis and Stearns [15, 29] and other started with their classification of languages and functions in terms of the time and space needed to compute them. Cobham [6] and Edmonds [11] in the 1960's introduced the notion of polynomial-time computation (see also previous work by Von Neumann [30]). Before Cook's and Levin's definition of the class NP, the P versus NP question appeared somewhat in the papers of Edmonds, Levin, Yablonski and in a letter from Gödel to Von Neumann [28].

In this paper I will give three different and equivalent formulations of the P vs. NP question, starting from the most natural one that connects it with verification algorithms. The second statement is in terms of the efficient solution of a particular problem and the last one uses the concept of Probabilistic Checkable Proofs, that is, probabilistic verification of a proof by checking a very small portion of it. I will also mention important connections of the P vs. NP problem with finite model theory and propositional proof systems. Finally, I will explore the ongoing research lines and the consequences of the two possible solutions of the problem. Notice that the mathematical consequence of P=NP, by using Occam razor principle, is that "we can then find proofs of theorems that have reasonably length proofs, say in under 100 pages. A person who proves P=NP would walk home from the Clay Institute not with one million-dollar check but with seven" (from Lance Fortnow in [13], see also [7]).

This historical introduction has been possible through detailed information provided in [7, 28, 27, 5].

Disclaimer. This chapter has been deliberately written for a nonexpert audience. For the purpose of clarity, formal and exact definitions have been often sacrificed. The references given intend to provide a minimal number of pointers for an interesting reader to start a more detailed study, and finally many important topics have been left out for space reasons but can be found in [3, 26].

2 Initial formulation of the problem

We start with the definition of the class P. A *decisional problem* is a problem in which each instance can have one of two possible solutions, that is, a Yes/No answer question such as "Given a natural number n, is n prime?" Formally a decisional problem $\Lambda = (I, R)$ is a set of data I codified over a finite alphabet, and a property R; the problem is stated as "Given x in I, does $R(x)$ hold?". In general problems, the solution to each input can take values in a larger set, for instance for the problem of computing the square root of a given number. Here we will only work with decisional problems, and will often drop the term "decisional".

We divide the set of all decisional problems in *complexity classes*, according to the resources used by an algorithm solving each problem. By the term *algorithm* we mean a finite set of instructions on any of a number of related models of computation, e.g., the Turing Machine or the Random Access Machine (this last one is an idealization of our everyday computers).

P is the class of problems that can be solved by an algorithm in time bounded above by a polynomial on the size of the input, that is, if a problem Λ is in P this means that

there are constants c, k such that the time needed for input x is at most $c|x|^k$, where $|x|$ is the size of the input, that is, the length of its codification. When we say time we mean the total number of steps taken by the algorithm. This definition of P is robust over the choice of a reasonable computation model.

An example of a problem in P is the boolean formula evaluation problem; given a well written boolean formula F with variables and boolean operators AND, OR, and NOT (such as the formula $(x \vee \neg y) \wedge z$), and given an assignment α that sets each of the variables as TRUE or FALSE (such as $x = y =$TRUE, $z =$FALSE for the above formula), does the formula F with assignment α evaluate to TRUE? (Usually when a formula F with an assignment α evaluates to TRUE we say that α satisfies the formula F). An algorithm for this problem working in polynomial time would be to first substitute each variable by the corresponding value and then simplify.

P is usually identified with the class of feasible problems, or problems that can be solved in practice. Evidently a polynomial-time bound can be huge both because of the multiplicative constant and the degree, but there are two practical reasons why we think of P as the efficiently solvable problems. Natural problems that are known to be in P have a very reasonable polynomial time bound, with degree at most 4, currently the extreme case is the problem of primality testing, very recently known to be solvable in polynomial time [1] for which the best known polynomial bounds have degree 6, but in this case I am convinced it is just a matter of time that this algorithm is improved. The second reason is that for natural problems not known to be in P, the best known algorithms take exponential time for some inputs, therefore the intermediate time bounds between polynomial and exponential don't seem to happen in practice.

The class NP can be defined in terms of verification algorithms or verifiers. A *verifier* V for a problem $\Lambda = (I, R)$ is an algorithm that takes two inputs, an input x in I and a *certificate* π. A certificate π is a candidate for a proof that $R(x)$ holds, that is, a witness of a Yes answer. The verifier V on input x, π outputs Yes if it can show that $R(x)$ holds using π, and No otherwise. The goal is that any Yes input has at least one certificate that convinces the verifier, whereas no certificate is valid for a No input. Formally, V verifies the problem Λ if for each x such that $R(x)$ holds there is a certificate π such that V with input x, π outputs Yes, and for every x such that $R(x)$ does not hold V outputs No on input x, π, for every π.

NP is the class of problems that can be verified by an algorithm running in time bounded by a polynomial on the size of its first input, that is, the input of the problem itself.

An example of a problem in NP is SAT or satisfiability of boolean formulae, that is, given a well written boolean formula F with operators AND, OR, and NOT, is there an

assignment α that satisfies F (that is, an assignment that makes the formula evaluate to TRUE)? A certificate for the verifier would be an assignment, and the algorithm only needs to check that the certificate satisfies the formula, which can be done in polynomial time as we have seen before.

It is easy to see that every problem in P is also in NP, the corresponding verification algorithm just ignores the certificate and then uses the polynomial time algorithm that solves the problem. So P$\subseteq$ NP, but does the other containment hold? Is P=NP? The question is thus a matter of solving vs. verifying a possible solution, we want to know if solving is really harder or less efficient than just verifying. We can also state our main question in terms of exhaustive search. It is clear that if a problem is in NP, then it can be solved by a brute force search of a certificate that satisfies the verifier, but such a procedure would take exponential time in many cases. Can we do better?, that is, can brute force search be replaced by an efficient (polynomial time) algorithm in all cases?

3 NP-completeness: a second formulation

In the previous section we gave a statement in terms of abstract problems, we want to know if each polynomial-time verifiable problem is also solvable in polynomial time. Here we have a formulation in terms of a particular problem, we see that P vs. NP is equivalent to the question of whether SAT, the Satisfiability problem, is in P.

In this approach we need the concept of a *reduction* between two problems. A problem A reduces to a problem B (denoted $A \leq_{\mathrm{m}}^{\mathrm{p}} B$) if there is a polynomial time algorithm that transforms each x, input of A, into an input of B, $f(x)$, that has exactly the same solution for B as x had for problem A. This means that if we have an efficient algorithm solving B we can combine it with a reduction from A to B and in this way we solve A efficiently. Notice that if B is in P and A reduces to B then A is also in P by the last argument. Equivalently, if A is not in P and A reduces to B then B is not in P.

We can therefore establish a partial ordering among the problems in NP by using the reductions $\leq_{\mathrm{m}}^{\mathrm{p}}$. Cook [9] proved in 1971 that the problem SAT is a maximum for this order, that is, every problem in NP reduces to SAT. Therefore it is enough to know whether SAT is in P. If SAT is in P then every problem that reduces to SAT is in P, therefore P=NP. If SAT is not in P then P$\neq$ NP, because SAT is in NP.

This property of SAT representing in a very strong sense the behaviour of the whole class NP holds for many other interesting problems. We say that a problem C is *complete* for NP, or NP-complete, if every problem in NP reduces to C. Therefore answering the question of whether C is in P, for a particular NP-complete problem C, would settle P vs. NP.

The list of known NP-complete problems includes important ones from virtually every area in science and engineering (see [14]) so in this sense the P vs. NP open question could be settled by a researcher from virtually any topic. After thirty years of unfruitful hard work in this direction, I don't think the analysis of the complexity of a particular problem will solve it. I explore in the next two sections alternative formulations that give a flavour of the robustness and strength of the problem statement and therefore of the foreseen difficulty of a solution.

4 Connections to logic: finite model theory and propositional proof systems

coNP is the class of complements of problems in NP, that is, A is in coNP if there is a problem B in NP such that x has solution Yes for A exactly when x has solution No for B.

If P=NP then clearly NP=coNP because the class P is closed under complementation of its problems. But the hypothesis NP $\neq$ coNP is stronger than P $\neq$ NP. It is plausible (although widely conjectured to be false) that P $\neq$ NP but NP=coNP.

In this section I will very briefly explore connections of the NP vs. coNP question and logic. The interested reader can find more details in [19] and [10, 21].

A *tautology* is a boolean formula that is true for any assignment of the variables. We denote as TAUT the set of all tautologies. A *propositional proof system* is a function f from proofs to tautologies that can be computed in polynomial time, where a *proof* is just a finite string of symbols. π is a *proof of* Ψ if $f(\pi) = \Psi$. An important question in propositional logic is whether there exists a propositional proof system such that every tautology has a polynomial size proof, that is, whether every true statement has a short proof that can be efficiently checked.

In fact it is known that it is enough to consider propositional proof systems that are based on modus ponens (formally, they are extensions of a SF, Frege system with substitutions). See [19] for all definitions.

Cook and Reckhow proved in [8] that NP=coNP is equivalent to the existence of a propositional proof system such that every tautology has a polynomial size proof.

In the rest of this section I will briefly sketch a different connection of logic and complexity theory. Fagin [12] proved that a problem $\Lambda = (I, R)$ is in NP if R can be written as an existential second-order formula with existential second order quantifiers.

I will not define first order or second order here (see for instance [10]) but I will give a few simple examples of their corresponding expressive power. Assume for a minute that our inputs are graphs (a graph is a set of vertices V and a binary relation E on V, the

edges). We can express with first order that a vertex u_0 is isolated:

$$\forall x \neg E(u_0, x) \wedge \neg E(x, u_0)$$

that is, the connectives $\wedge, \vee, \neg$ are allowed, as well as universal and existential quantifiers on vertices. I can express 2-colorability, that is, there is a way to colour all vertices with two colours such that adjacent vertices don't have the same colour as follows:

$$(\exists P)(\forall x)(\forall y)(E(x, y) \rightarrow (P(x) \leftrightarrow \neg P(y)))$$

Notice that I have quantified over relations here, that is, I am using second order logic. The reader can try to express 3-colorability with second-order.

More recently, P has also been characterized using the expressive power of a more sophisticated logic, first order with fixed point operators. The fixed point operator allows us to iterate the relation "there is an edge from vertex u to vertex v" to "vertices u and v are connected (by a path of any length)"

$$\begin{aligned} \phi^1(x, y) &\equiv E(x, y) \\ \phi^{m+1}(x, y) &\equiv \phi^m(x, y) \vee \exists z(E(x, z) \wedge \phi^m(z, y)) \end{aligned}$$

The iteration of this process, ϕ^∞, is the fixed point of E.

The question of whether P=NP is thus equivalent to a question on logic expressibility, namely whether existential second order and first order with fixed point express the same properties.

5 Probabilistically Checkable Proofs

In this section we characterize the class NP with a generalization of verification algorithms called Probabilistically Checkable Proofs. This result was proven in 1992 [2] and received a wide attention from the Computational Complexity community for two reasons. On the one hand this definition of NP had dramatically different properties from the previously known characterizations, in the sense that this one does not relativize as I will explain below. On the other hand many negative approximation results were derived from it, meaning that it was proven that no approximated solution to many optimization problems exists unless P=NP.

We consider probabilistic algorithms, that is, algorithms that have access to random bits, which means that at any point of its execution the algorithm can request a random bit and receive it in unit time, the bit being 1 with probability 1/2.

We define probabilistic verifiers which are verifiers such as those defined in section 2 (that is, the input of the verifier is the original input x and a certificate π) but with the additional power of probabilistic algorithms.

So now the verification algorithm on a fixed input x and certificate π can give different outputs because they depend on the random bits received, so we need to relax the notion of a verifier being correct for a problem.

A probabilistic verifier V is valid for a problem $\Lambda = (I, R)$ if for each input $x \in I$, if $R(x)$ holds then there is a certificate π such that V outputs Yes with probability 1 (the probability is taken over the random bits produced); if $R(x)$ does not hold then for any certificate π, the probability of V giving output Yes is smaller than 0.1

$$\begin{aligned} R(x) &\Rightarrow \exists \pi \Pr(V(x, \pi)) = 1 \\ \text{NOT } R(x) &\Rightarrow \forall \pi \Pr(V(x, \pi)) < 0.1 \end{aligned}$$

Notice that for Yes inputs there is a certificate for which the verifier always gives a correct answer, whereas for No inputs and any certificate the output can be wrong with a small probability.

Polynomial-time probabilistic verifiers are very powerful if we allow them to use polynomially random bits and to have full access to the certificate, in fact they correspond in this case to exponential time verifiers and the complexity class NEXP, that is, the exponential time analogous to nondeterministic polynomial time NP.

This is the reason why we introduce two parameters restricting the amount of randomness and the access to the certificate. The certificate π can be read one symbol at a time by requesting the symbol in a particular position of π and getting it in unit time, that is, a direct access mechanism. We can now restrict the number of symbols in the certificate that are actually read.

It is clear that if we only restrict the number of random bits use to 0 we get exactly the class NP, because with that restriction only we get our original polynomial time verifiers. But what happens if we use randomness? Can we probabilistically verify without having to read the whole certificate? This would correspond to the idea of quickly checking (very long) candidate proofs of a theorem; correct proofs should be accepted but there is a small probability of accepting a false proof.

From 1990 to 1992 several results appeared in this line, first showing that a polylogarithmic number of both random bits and certificate access were sufficient to capture NP, and then getting conditions that were both sufficient and necessary. See for instance [23] for the whole story. The best known result [2] is that NP is exactly the class of problems that can be solved by probabilistic verifiers using a constant number of certificate bits (that is, the same number of bits for all inputs and certificates) and a logarithmic number of random bits, this is the complexity class $\mathrm{PCP}(\log n, 1)$.

These probabilistic verifiers are thus very restricted, they check a very small amount of the certificate and get a correct answer with high probability. The result was proven for the NP-complete SAT, by a beautiful arithmetization technique that transforms each boolean assignment into a linear function. Our P versus NP problem is now transformed into the question of whether logarithmically many random bits and a constant number of times of certificate access can do more than a regular polynomial-time algorithm.

This is the first known form of the problem that does not relativize, which was very celebrated by the researchers in Computational Complexity. Why were they so happy? Assume that we live in a world were the solution of a particular problem A is given for free. This means that at any point an algorithm can ask for the solution of A on a particular input y an get an exact solution in unit time. This is called having A as an oracle. We can now define the class of polynomial-time solvable problems in this world A, denoted as P^A, and similarly for verifiers. The standard separation techniques that were used in attacking the P versus NP question were all known to relativize, meaning that they only give results that are independent of the oracle, that is, results that hold in any possible world A. But this is useless because it is known that there are oracles for which $\mathrm{P}^A = \mathrm{NP}^A$ and other for which $\mathrm{P}^A \neq \mathrm{NP}^A$, so techniques that relativize will never solve the question. The equality NP=PCP$(\log n, 1)$ is known not to hold for some oracles and therefore proofs that rely on this characterization of NP do not relativize, so there is some hope that they can obtain stronger results than those of known relativizable techniques.

6 Research directions

In this section we list research areas created and/or highly motivated by the P vs. NP question.

Computational Complexity [3] defines different complexity classes such as P and NP in terms of different computing resources. The main open question in this context is the separation of different complexity classes, that is, whether they have the same problems. Typical tools are reductions, such as $\leq_{\mathrm{m}}^{\mathrm{p}}$, that can vary according to the resources used in the computation of the reduction itself and the access the reduction gives to the problem it reduces to, for instance in the case of $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ the reduction accesses a single input of the problem B and the solution to this input is exactly the solution for the original input of A.

Besides the more specific directions we mention below, there is a rich variation of techniques in Computational Complexity, starting with the classical diagonalization and counting techniques and including quantitative approaches such as resource-bounded measure [22] and highly nonclassical ones such as quantum computing complexity classes [24].

An important question in complexity is whether having access to a source of random bits can make computation substantially quicker. There are several **probabilistic complexity classes** corresponding to different allowed computation errors in this context, for instance one-sided or two-sided errors for the case of decisional problems. BPP is the class of problems that can be solved in polynomial time with a source of random bits and allowing an exponentially small on both possible cases, Yes and No instances. It is open whether BPP=P and in this case a positive answer is not ruled out by many complexity theorists [25].

Average case complexity considers a probability distribution on the inputs of a problem and measures the time needed to solve it as an average value, as opposed to the worse case complexity we use in the definition of P, for instance, where we consider the time needed for the slowest input of each length. We are not looking for algorithms that solve quickly all instances of a problem but for those that work well on average [31].

The size of **Boolean circuits** [32, 3] that solve a problem is another complexity measure. This is a nonuniform computational model, since a different circuit is needed for each input size. The advantage of these simple models is that lower bounds have been obtained, at least for small bounds, and it seems plausible that this work direction can give more powerful results (not with the known techniques though). It is known that each problem in P has polynomial-size circuits so showing that a single problem in NP does not have polynomial size circuits would separate these two classes.

Approximation algorithms is a very active area of research that has used the PCP characterization of NP (see section 5) for negative results. They consider the optimization problems corresponding naturally to many NP-complete problems, for example MAXCLIQUE is the problem of computing the size of the maximum complete subgraph, which is the optimization problem corresponding to NP-complete CLIQUE (CLIQUE is the problem of deciding whether a graph has a complete subgraph of a given size).

The behaviour of these NP-complete based optimization problems is very varied in terms of how well they can be approximately solved. Some of them can be solved in polynomial time with an exponential error whereas for other just solving the problem with a constant error would imply P=NP.

Descriptive complexity [17] explores the characterization of complexity classes in terms of logic expressibility, in the line of Fagin's characterization of NP ([12], section 4). Immerman [16] characterized P, NL (nondeterministic logarithmic space) and other complexity classes and proved that NL=coNL using logic techniques. There is a parallel line of research dealing with algebraic characterizations of complexity classes [4].

Proof complexity [19] explores the connection we introduced in section 4 between

NP=coNP and the existence of short proofs for tautologies. The idea is to prove superpolynomial lower bounds for the length of proofs in propositional proof systems of increasing complexity, in order to end up obtaining the result for every propositional proof system.

7 Consequences

Most complexity researchers believe that P is different from NP. In fact much more is expected to hold, the assumptions used in cryptography include that integer factoring cannot be done in polynomial time, which implies P≠NP, and in fact it is even assumed (for instance in DES) that factoring cannot be done in polynomial time for "many" integers. Since multiplication is feasibly computable, factoring can be formulated as inverting a polynomial time computable function. The existence of a polynomial time computable function that cannot be inverted in polynomial time is crucial in modern cryptography and is conjectured to be a much stronger hypothesis than P≠NP.

As explained above, a feasible algorithm for an NP-complete problem (therefore showing P=NP) would mean the end of DES and most currently used cryptographic protocols, with devastating financial and military consequences. But not all consequences would be negative. Consider the problem X of deciding whether an input T, p corresponds to a valid theorem T and a prefix of a formal proof of T, p, where a formal proof is detailed enough to be checked by a computer. This problem X is in NP, so if P=NP it can be solved in polynomial time, and this would mean a breakthrough in mathematics. For any theorem which has a proof of reasonable length we can efficiently find such a proof!

Although I strongly believe this is not the case (I am sure that P is different from NP), the possibility of walking home with all seven Clay Institute prize checks, as Lance Fortnow says in [13], is definitely an incentive for the other direction.

References

[1] M. Agrawal, N. Kayal, and N. Saxena. Primes is in P, 2002.

[2] 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):501–555, 1998.

[3] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I (second edition)*. Springer-Verlag, Berlin, 1995.

[4] D. A. Mix Barrington and D. Thérien. Finite monoids and the fine structure of NC1. *Journal of the ACM*, 35:941–952, 1988.

[5] Daniel Pierre Bovet and Pierluigi Crescenzi. *Introduction to the Theory of Complexity.* Prentice Hall, 1994.

[6] A. Cobham. The intrinsic computational difficulty of functions. In *Proceedings of the 1964 International Congress for Logic, Methodology, and Philosophy of Science*, pages 24–30, 1964.

[7] S. Cook. The P versus NP problem. Manuscript.

[8] S. Cook and R. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44:36–50, 1979.

[9] S. A. Cook. The complexity of theorem proving procedures. In *Proceedings of the Third ACM Symposium on the Theory of Computing*, pages 151–158, 1971.

[10] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer-Verlag, 1999.

[11] J. Edmonds. Minimum partition of a matroid into independent subsets. *J. Res. Nat. Bur. Standards Sect. B*, 69:67–72, 1965.

[12] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *Complexity of Computation, SIAM-AMS Proceedings*, 4:43–73, 1974.

[13] L. Fortnow. My computational complexity web log, may 25, 2004.

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* W.H. Freeman and Company, San Francisco, 1979.

[15] J. Hartmanis and R.E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[16] N. Immerman. Languages that capture complexity classes. *SIAM Journal on Computing*, 16:760–778, 1987.

[17] N. Immerman. *Descriptive Complexity.* Springer-Verlag, 1999.

[18] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–104. Plenum Press, 1972.

[19] J. Krajicek. *Bounded Arithmetic, Propositional Logic and Complexity Theory.* Cambridge University Press, 1996.

[20] L. A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9:265–266, 1973.

[21] L. Libkin. *Elements of Finite Model Theory.* Springer-Verlag, 2004.

[22] J. H. Lutz. The quantitative structure of exponential time. In L. A. Hemaspaandra and A. L. Selman, editors, *Complexity Theory Retrospective II*, pages 225–254. Springer-Verlag, 1997.

[23] E. W. Mayr, H. J. Prmel, and A. Steger. *Lectures on Proof Verification and Approximation Algorithms*, volume 1367 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[24] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[25] N. Nisan and A. Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.

[26] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[27] J.E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison-Wesley, 1998.

[28] M. Sipser. The history and status of the P versus NP question. In *Proceedings of the 24th Annual ACM Symposium on the theory of Computing*, pages 603–618, 1992.

[29] R. E. Stearns, J. Hartmanis, and P.M. Lewis. Hierarchies of memory limited computations. In *Proc. 6th Annual Symp. on Switching Circuit Theory and Logical Design*, pages 179–190, 1965.

[30] J. von Neumann. A certain zero-sum two-person game equivalent to the optimal assignment problem. In H.W. Kahn and A.W. Tucker, editors, *Contributions to the Theory of Games II*. Princeton University Press, 1953.

[31] J. Wang. Average-case computational complexity theory. In L. Hemaspaandra and A. Selman, editors, *Complexity Theory Retrospective II*, pages 295–328. Springer-Verlag, 1997.

[32] I. Wegener. *The Complexity of Boolean Functions*. John Wiley & Sons, 1987.