ARBOR Ciencia, Pensamiento y Cultura
Vol. 189-764, noviembre-diciembre 2013, a080 | ISSN-L: 0210-1963
doi: http://dx.doi.org/10.3989/arbor.2013.764n6003

EL LEGADO DE ALAN TURING / *THE LEGACY OF ALAN TURING*

TURING'S ALGORITHMIC LENS: FROM COMPUTABILITY TO COMPLEXITY THEORY

LA LENTE ALGORÍTMICA DE TURING: DE LA COMPUTABILIDAD A LA TEORÍA DE LA COMPLEJIDAD

Josep Díaz* and Carme Torras**
*Llenguatges i Sistemes Informàtics, UPC.
diaz@lsi.upc.edu
**Institut de Robòtica i Informàtica Industrial, CSIC-UPC.
torras@iri.upc. edu

Citation/Cómo citar este artículo: Díaz, J. and Torras, C. (2013). "Turing's algorithmic lens: From computability to complexity theory". *Arbor, 189* (764): a080. doi: http://dx.doi.org/10.3989/arbor.2013.764n6003

Received: 10 July 2013. Accepted: 15 September 2013.

ABSTRACT: The decidability question, i.e., whether any mathematical statement could be computationally proven true or false, was raised by Hilbert and remained open until Turing answered it in the negative. Then, most efforts in theoretical computer science turned to complexity theory and the need to classify decidable problems according to their difficulty. Among others, the classes **P** (problems solvable in polynomial time) and **NP** (problems solvable in non-deterministic polynomial time) were defined, and one of the most challenging scientific quests of our days arose: whether **P = NP.** This still open question has implications not only in computer science, mathematics and physics, but also in biology, sociology and economics, and it can be seen as a direct consequence of Turing's way of looking through the algorithmic lens at different disciplines to discover how pervasive computation is.

KEYWORDS: Computational complexity; the permanent problem; P and NP problems; Interactive proofs and holographic proofs.

RESUMEN: La cuestión de la decidibilidad, es decir, si es posible demostrar computacionalmente que una expresión matemática es verdadera o falsa, fue planteada por Hilbert y permaneció abierta hasta que Turing la respondió de forma negativa. Establecida la no-decidibilidad de las matemáticas, los esfuerzos en informática teórica se centraron en el estudio de la complejidad computacional de los problemas decidibles. En este artículo presentamos una breve introducción a las clases **P** (problemas resolubles en tiempo polinómico) y **NP** (problemas resolubles de manera no determinista en tiempo polinómico), al tiempo que exponemos la dificultad de establecer si **P** = **NP** y las consecuencias que se derivarían de que ambas clases de problemas fueran iguales. Esta cuestión tiene implicaciones no solo en los campos de la informática, las matemáticas y la física, sino también para la biología, la sociología y la economía. La idea seminal del estudio de la complejidad computacional es consecuencia directa del modo en que Turing abordaba problemas en diferentes ámbitos mediante lo que hoy se denomina la lupa algorítmica. El artículo finaliza con una breve exposición de algunos de los temas de investigación más actuales: transición de fase en problemas **NP**, y demostraciones holográficas, donde se trata de convencer a un adversario de que una demostración es correcta sin revelar ninguna idea de la demostración.

PALABRAS CLAVE: Complejidad computacional; el problema de calcular la permanente de una matriz; problemas P y NP; demostraciones interactivas y holográficas.

1. INTRODUCTION

The English mathematician Alan Mathison Turing (1912-1954) is well-known for his key role in the development of computer science, but he also made important contributions to the foundations of mathematics, numerical analysis, cryptography, quantum computing and biology. In the present paper we focus exclusively on the influence of Turing on *computability and complexity theory,* leaving aside other aspects of his work even within computer science, most notably his contribution to the birth of the digital computer and artificial intelligence.

Turing was an important player in the settlement of the decidability issue, which led naturally to the study of the complexity of decidable problems, i.e., once it was proved that there are problems unsolvable by a computer, research turned to the question of "how long would it take to solve a decidable problem?" This is an exciting research field with lots of open questions. In this paper, we try to give a gentle introduction to the historical perspective of the search for efficient computing and its limitations.

2. BEFORE TURING: THE DREAM OF MECHANICAL REASONING AND DECIDABILITY

Gottfried W. Leibniz (1646-1716) was an important mathematician, philosopher, jurist and inventor, whose dream was to devise an automatic reasoning machine, based on an alphabet of unambiguous symbols manipulated by mechanical rules, which could formalize any consistent linguistic or mathematical system. He produced a calculus ratiocinator, an algebra to specify the rules for manipulating logical concepts. More than a century later, George Boole (1815-1864) and Augustus De Morgan (1806-1871) converted the loose ideas of Leibniz into a formal system, Boolean algebra, from which Gottlob Frege (1848-1925) finally produced the fully developed system of *axiomatic predicate logic.*

Frege went on to prove that all the laws of arithmetic could be logically deduced from a set of axioms. He wrote a two-volume book with the formal development of the foundations of arithmetic, and when the second volume was already in print, Frege received the celebrated letter from Bertrand Russell (1872-1970) showing his theory was inconsistent. The counterexample was the paradox of *extraordinary* sets. A set is defined to be *extraordinary* if it is a member of itself, otherwise it is called *ordinary. Is the set of all ordinary sets also ordinary?* (see Doxiadis, Papadimitriou, Papadatos and di Donna, 2009, pp. 169 to 171 for a particular charming account of the effect of Russell's letter to Frege).

Frege's work was the spark for 30 years of research on the foundations of mathematics, and the end of the 19th c. and beginning of the 20th c. witnessed strong mathematical, philosophical and personal battles between members of the *intuitionist* and *formalist* European schools (Davis, 2000; Doxiadis, Papadimitriou, Papadatos and di Donna, 2009).

To better frame further developments, let us recall some notions. A *formal system* is a language, a finite set of axioms, and a set of inference rules used to derive expressions (or statements) from the set of axioms; an example is mathematics with the first-order logic developed by Frege. A system is said to be *complete* if every statement can be proved or disproved; otherwise the system is said to be *incomplete.* A system is said to be *consistent* if there is not a step-by-step proof within the system that yields a false statement.

A system is said to be *decidable* if, for every statement, there exists an algorithm to determine whether the statement is true.

In 1928, at the International Congress of Mathematicians in Bologna, David Hilbert (1862-1943), a leading figure in the formalist school, presented to his fellow mathematicians three problems, the second of which would have a big impact on the development of computer science: the *Entscheidungsproblem.* In English, it is named the *decision problem,* and it can be formulated as follows: "provide a method that, given a first-order logic statement, would determine in a finite number of steps whether the statement is true". In other words, the Entscheidungsproblem asks for the existence of a finite procedure that could determine whether a statement in mathematics is true of false. Hilbert was convinced that the answer would be yes, as mathematics should be complete, consistent and decidable.

3. TURING ESTABLISHES THE LIMITS OF COMPUTABILITY

In 1936 Turing completed the draft of his paper "On Computable Numbers, with an Application to the Entscheidungsproblem", where he proposed a formal model of computation, the *a-machine,* today known as the *Turing machine* (TM), and he proved that any function that can be effectively computed (in Hilbert's sense) by a human being could also be mechanically calculated in a finite number of steps by a mechanical procedure, namely the TM. Turing used his formalism to define the set of *computable numbers,* those reals for which their *i*th decimal can be computed in a finite number of steps. The number of computable numbers is countable but the number of reals is uncountable, so most of the reals are not computable. Some well-known computable numbers are 1/7, π and *e*.

In his model, Turing introduced two essential assumptions, the discretization of time and the discretization of the state of mind of a human calculator. A TM consisted of an infinite tape divided into squares, a finite input on a finite alphabet, and a write-read head that could be in a finite number of states representing those of the human calculator's brain (Davis, 2000; Moore and Mertens, 2011). The TM was a theoretical model of a *stored-in program* machine. An outstanding achievement of Turing was the concept of *Universal Turing Machine* (UTM). Given the codified description of a TM as input, the UTM is able to simulate the computation of that machine and write the result. Hence, the UTM can simulate any other Turing machine, thus opening the way to digital computers.

Relying on his UTM, Turing made a decisive contribution to computability, namely he produced a problem that was undecidable: the *halting problem.* It can be stated as follows: "Given a codified description of a TM and an input to it, decide if the UTM will halt". The proof was a mimic of Geodel's incompleteness one, where a diagonalization argument produces a paradox (see Davis, 2000, chapter 7 for a nice informal explanation). Note that the halting problem is a counterexample, a negative solution to the Entscheidungsproblem[1].

Turing's PhD. thesis extended Gödel's incompleteness theorem by proving that when new axioms are added to an incomplete formal system the system remains incomplete (Turing, 1939). A very important contribution of that work was the *Oracle Turing-machine* model. In the words of Turing: "Let us suppose we are supplied with some unspecified means of solving number-theoretical problems, a kind of oracle as it were. We shall not go any further into the nature of this oracle apart from saying that it can't be a machine".

Emil Post (1897-1954) realized that Turing's oracle machine had important implications for computability, as it could be used to compare the relative difficulty of problems; thus he defined the concept of Turing-reducibility (T-reducibility). Given problems P_1 and P_2, P_1 is said to be *Turing-reducible to* P_2 ($P_1 \leq_T P_2$) *if* P_2 can be used to construct in a finite number of steps, a program to solve P_1. Note that if $P_1 \leq_T P_2$ and P_2 is decidable then P_1 is also decidable, and if $P_1 \leq_T P_2$ and P_1 is undecidable, then P_2 is also undecidable. Therefore, T-reducibility can be used to enlarge the class of know undecidable problems.

4. AFTER TURING: PROBLEM COMPLEXITY CLASSES

We have seen that the work of Turing and others let us determine, for any given problem, whether there is an algorithm to solve it (i.e., the problem is decidable) or not. Let us turn into the realm of solvable problems, and ask the question of how long it takes to solve a given decidable problem. It may seem that this is a pure technological issue, however we will see that there are problems for which today's computers would take more that the age of universe to find a solution, when the input to the problem is a bit large.

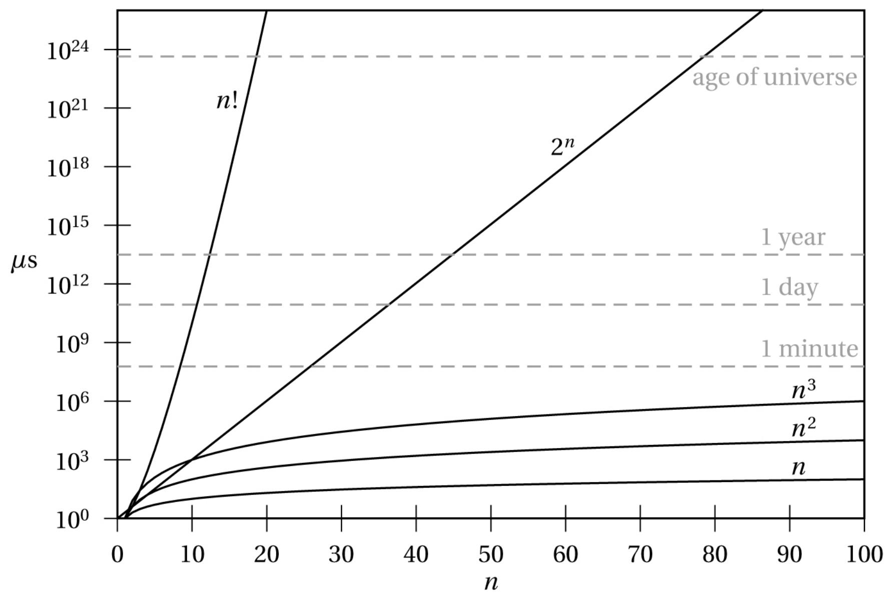
The *time complexity* of a problem with input x is a measure of the number of steps that an algorithm will take to solve it, as a function of the size of the input $|x|$ = n. Let us consider the *worst-case complexity,* where among all the possible inputs of size *n*, we take the one that behaves worst in terms of the number of steps performed by the algorithm. As an example, consider the school algorithms for multiplying two integers *a* and *b* of sizes n_a and n_b, respectively, size measured as the number of digits in the binary expression (i.e., $n_a = \log_2 a$), and let $n = \max\{n_a, n_b\}$, then the worst-case complexity of the algorithm is $T(n) = O(n^2)$.[2]

For decidable problems, the time complexity expression has a tremendous impact on the running time of an algorithm. Figure 1 (taken from Moore and Mertens, 2011) is an indication of the running time of different worst-case complexity figures, assuming a processor can solve an instance of size $n = 1$ in 10^{-3} seconds, which is consistent with today's technology. Note that an algorithm that solves a problem with complexity $O(2^n)$, if given an input of size $n = 90$, may yield the output after the universe would be finished (again, assuming today's technological level).

Let us try to get a feeling for the time-complexity of some problems. Given an n x n matrix $M = (m_{ij})$, its *determinant* is defined by the Leibniz formula

$$\det(M) = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} m_{i,\sigma_i} m_{i,\sigma(i)},$$

Figure 1: Running times as a function of input size n



where σ is one permutation from the symmetric group of permutations S_n, and sgn (σ) is the sign of the permutation. A similar associate value of a square matrix is the *permanent,* which is defined

$$\mathrm{perm}(M) = \sum_{\sigma\in S_n}\prod_{i=1}^{n} m_{i,\sigma_i} m_{i,\sigma(i)}.$$

For example if

$$M = \begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix}$$

then det(M) = $m_{11} \cdot m_{22} - m_{12} \cdot m_{21}$, while perm($M$) = $m_{11} \cdot m_{22} + m_{12} \cdot m_{21}$.

A direct implementation of the definition yields an $O(n!)$ algorithm for both problems. However, by using the LU decomposition by Gauss and Turing, the determinant of an n x n matrix can be computed in $O(n^3)$ steps, see for example in Cormen, Leiserson, Rivest and Stein (2001), while the complexity of computing the permanent seems to be much more difficult (Valiant, 1979). Recently D. Glynn has obtained a deterministic bound of $O(n2^n)$ to the complexity of the permanent (Glynn, 2010). Notice that if M_1 and M_2 are square matrices then det(M_1M_2) = det(M_1) x det(M_2), while perm(M_1M_2) ≠ perm(M_1) x perm(M_2).

A problem is said to be *feasible* if there exists an algorithm that solves it with polynomial time-complexity. All other decidable problems are said to be *unfeasible,* which does not imply that in the near or distant future some problems may turn into feasible.

A particularly interesting kind of problems are the *combinatorial optimization problems,* where for any instance x and a solution s_x for that instance, there is a cost function $k(x, s_x) \in \mathbb{R}^+$ and the objective is to find the s_x^* that optimizes (maximizes or minimizes) $k(x, s_x)$ over all possible solutions s_x[3]. Let Opt(x) denote the cost of an optimal solution.

For example, consider the *chromatic number problem* of a graph: "Given as input a graph $G = (V, E)$, find the minimum number of different colors needed to have a valid coloring of the vertices in V", where a valid coloring of G is an assignment of labels $\chi : V \rightarrow \{1, 2, ..., k\}$, each integer representing a color, such that for any $(u, v) \in E$, $\chi(u) \neq \chi(v)$. Notice that, for any input G and solution χ, Opt(G) is a valid coloring with minimum number of colors.

4.1 The classes P, NP and NP-complete

In the early 1950's, some mathematicians started to realize that some decidable problems took too long to be solvable for large inputs. In a series of letters to the US National Security Agency, John Nash proposed a secure cryptographic encryption system based on the computational difficulty of problems (Nissan, 2004).

These letters foresaw the classification of decidable problems according to their computational complexity and could have given birth to the field of *complexity theory* if the letters had not remained a state secret until 2012.

The second important historical event for the study of problem complexity was the letter Kurt Gödel sent to John von Neumann in March 1956. At the time, von Neumann was dying and he did not read the letter, which was lost until the 1980's. An English translation of the letter can be found in the appendix of Lipton's book (2010), which is a compilation of selected posts of his useful blog. As it is beautifully explained in Lipton's poetic version of the events, "Kurt Gödel is walking along through the falling snow, he is thinking. Gödel is the greatest logician of his time, perhaps of all times, yet he is deeply troubled. He has found a problem he cannot solve. The problem concerns a simple but fundamental question. Suddenly he smiles, he has an idea. If he cannot solve the problem, then he will write a letter explaining it to John von Neumann; perhaps John will be able to solve the problem. After all John von Neumann has one of the fastest minds in the world. Gödel pulls his scarf tighter, gets his bearings in the heavy falling snow, and heads to his office to write his letter".

In his letter, Gödel considered the *truncated Entscheidungsproblem problem:* "Given any statement in first-order logic (for example $\exists x, y, z, n \in \mathbb{N} - \{0\} : (n \geq 3) \wedge (x^n + y^n = z^n)$) decide if there is a proof of the statement with finite length of at most *m* lines, where *m* could be any large (10^{10}) constant. As any mathematical proof could be represented using a small constant number *c* of symbols, the problem is decidable; just construct all the exponentially many c^m possible proofs of length *m*, over the finite alphabet, and check if any of them works. Notice that most of the proofs will be gibberish. Moreover, for large *m*, there is a chance that the process could take longer than the existence of the earth! (see Fig. 1). Gödel in his letter asked if it was possible to do it in $O(m^2)$ steps, making explicit the possible partition between decidable problems solved in polynomial time, and decidable problems that cannot be solved in polynomial time.

During the 1960's, the existence of easy and hard problems started to be more obvious to researchers. As computers were solving problems with increasing input size, researchers begin to consider the effect of input size on the computation time (number of steps) and space (number of cells visited), using the Turing machine as the computational model. An important contribution appeared in 1965 by Hartmanis and Stearns (1965), which defined the multi-tape Turing machine and laid the foundations of complexity theory. The same year, Edmonds (1965) gave an informal description of *non-deterministic polynomial-time problems,* i.e., those that permit verifying in polynomial time whether a guessed hypothetical solution is indeed a correct solution to the problem. Nondeterminism would play a key role in the development of computational complexity theory. A nondeterministic algorithm could be seen as a game where a computationally omnipotent *prover* $\mathcal{P}$ provides a *witness* to the solution of the problem, and a *verifier* has to prove deterministically that the witness is indeed a correct solution.

Consider the problem of the *satisfiability* of a conjunctive normal form (SAT): "Given a set of Boolean variables *X* and a Boolean formula Φ on X in conjunctive normal form, decide if there is an assignment A : $X \rightarrow \{T, F\}$ (truth, false) such that it satisfies Φ" (evaluates Φ to T). A conjunctive normal form formula consists of the conjunction of clauses, where each clause is a disjunction of literals (Boolean variables or their negation). For example, if $X = \{x_1, x_2, x_3, x_4\}$, consider the Boolean formula

$$\Phi = (x_1 \vee \bar{x_2} \vee x_4) \wedge (\bar{x_1} \vee \bar{x_2} \vee \bar{x_4}) \\ \wedge (x_1 \vee \bar{x_3} \vee \bar{x_4}) \wedge (\bar{x_1} \vee x_2 \vee \bar{x_3})$$

A backtracking deterministic algorithm would find an assignment satisfying Φ. In a nondeterministic algorithm, $\mathcal{P}$ would supply as witness an assignment A (for example, $A(x_1) = A(x_4) = T, A(x_2) = A(x_3) = F$) and $\mathcal{V}$ would have to verify that the substitution of such an assignment makes Φ satisfiable. Note that if Φ has input size *n* (length of Φ), $\mathcal{V}$ could verify whether the assignment satisfies Φ in *O*(*n*) steps.

In 1971 Stephen Cook showed that the SAT problem was a paradigmatic unfeasible problem in the sense that any problem that could be solved[4] in polynomial time by a nondeterministic Turing machine could be *reduced* to the SAT problem, where the reducibility is the same concept as Turing-reducibility but imposing the condition that the construction should be done in polynomial time (Cook, 1971). The paper was presented at the STOC-71 conference, and again using the Lipton (2010) description: "A tall figure walks slowly to the front of the conference room. Steve Cook is a young scientist, who is about to change the world. He has independently discovered the problem that troubled Gödel that snowy day. He gives his talk, and after there is a polite applause, as there is for every talk".

One attendee who understood perfectly the meaning of Cook's result was Richard Karp, who the following year published a seminal paper formally defining the classes **P**, **NP** and **NP**-complete, and provided the proof that nine well-known problems were in the class **NP**-complete (Karp, 1972). His starting seed was that SAT ∈ **NP**-complete, which he coined as Cook's Theorem. At the same time, but independently from Cook and Karp, a Russian mathematician, Leonid Levin, was basically defining the class **NP**-complete (Levin, 1973), the article appeared translated into English the same 1973, but it took a while for the community working in complexity theory to realize the meaning of Levin's result (see Trakhtenbrot, 1984 for a history of early complexity theory developments in Russia). Today the result that SAT is **NP**-complete is known as the Cook-Levin Theorem.

We next give an intuitive description of the complexity classes. For a more technical description the reader is referred to any of the complexity or algorithmic books listed in the bibliography of this paper.

P is the class of problems for which there is a deterministic algorithm finding a solution in polynomial time, for any input.

NP is the class of problems such that if an omniscient prover $\mathcal{P}$ provides a polynomial-length witness to the solution of the problem, a verifier $\mathcal{V}$ can prove *in polynomial time* whether the witness is indeed correct. The term **NP** stands for *non-deterministic polynomial time.*

From the previous remarks on the SAT problem, it is clear that SAT $\in$ **NP**: $\mathcal{P}$ supplies a valid assignment and $\mathcal{V}$ checks in linear time that the assignment satisfies the formula. However, if we consider a combinatorial optimization problem, as the chromatic number of a graph, it is not so easy. Given an input $G = (V, E)$, the *chromatic number problem* consists of finding a *minimum* valid coloring χ* for G", i.e., assigning the minimum number of colors to V such that for every edge $(u,v) \in E$, $\chi^* (u) \neq \chi^* (v)$. To prove this problem is not in **NP,** assume that $\mathcal{P}$ provides a witness χ* to the colorability of *V,* then $\mathcal{V}$ tests that indeed χ* is a valid coloring by looking at every edge of *G,* which has a cost of $O(n^2)$. Moreover, $\mathcal{V}$ must also verify that there is no other valid coloring with less colors that χ*, i.e., he must compare with all other possible valid colorings for *G,* which could be exponential. Therefore, the problem is not in **NP.**

To circumvent this difficulty, we consider the *search version* of the combinatorial optimization problem, where as a part of the input we also are given a value *b* that the cost function can take. If the optimization problem demands to maximize, the search version would require that $k(x) \geq b$, and if it is a minimization problem the search version would require $k(x) \leq b$.

Continuing with the example, the search version of the *chromatic number problem* can be stated as follows: "Given as input $G = (V, E)$ and a $b > 0$, find a valid coloring χ of G such that $|\chi(V)| \leq b$".

The search versions of optimization problems are obviously in **NP.** Furthermore, by using binary search it is a standard exercise to prove that if the search version of an optimization problem can be solved in polynomial time then the optimization version can also be solved in polynomial time (see Exercise 8.1 in Dasgupta, Papadimitriou and Vazirani, 2008).

Let us see some further examples of problems in **P** and **NP**:

The *primality problem:* "Given an integer a with length $n = \log a$ bits, decide whether *a* is prime". There is a non-trivial proof that primality ∈ **NP** due to V. Pratt (1975), and for a long time it was open whether the problem was in **P**. In 2002, Agrawal, Kayal and Saxena provided a deterministic polynomial-time algorithm to solve the primality problem, so the problem is in **P** (Agrawal, Kayal, and Saxena, 2002).

The *factorization problem:* "Given an integer *a* of *n* bits, find its prime decomposition". This is an important problem as, among other things, it is the basis of the *RSA,* one of the most used public-key cryptographic systems (see Chapter 8 in Singh, 2000). Factorization is in **NP,** since if $\mathcal{P}$ provides a witness $p_1,\dots,p_m$, $\mathcal{V}$ can test that each p_i is prime (using the algorithm in Agrawal, Kayal, and Saxena, 2002) and see that the product is the given number. Nevertheless, it is an important open question whether there is a polynomial-time algorithm to solve factorization[5].

The problem of *3-satisfiability* (3-SAT) is the particular case of SAT where every clause in the input has exactly 3 literals. The problem is also in **NP.** However, an easy greedy algorithm puts *2-satisfiability* in the class **P.**

Finally, the *truncated Entscheidungsproblem* is also in **NP,** since if we get a proof of less than 10^{10} pages as a certificate, any mathematician can verify in a reasonable time (polynomial in 10^{10}) whether it is correct (independently that there exist other valid proofs among the exponential number of generated ones).

Notice that **P** ⊆ **NP**. If $\mathcal{V}$ can obtain a deterministic solution in polynomial time, he can also verify it in polynomial time.

The problem to decide whether **P=NP** was considered one of the nine millennium problems posed by the *Clay Mathematics Institute* in the year 2000. A 10^7 US$ prize is awarded for each solution to one of the problems.

The answer **P≠NP** would mean that, for many important search problems, *finding* is more difficult than *verifying.* If, on the contrary, the answer is **P=NP,** then there would be a feasible algorithm for the truncated Entscheidungsproblem, i.e., proofs with a large but finite number of lines, which in practice would suffice for most mathematical theorems.

4.2 Karp reducibility and NP-completeness

To further study the relationship between search problems and the **P=NP** question, Karp defined the following simplified variation of Turing-reducibility: Given search problems A and *B,* with sets of instances I_A and I_B and sets of solutions S_A and S_B, a *Karp reduction* from *A* to *B* ($A \leq_P B$) is a polynomial-time computable function $f: I_A \rightarrow I_B$ such that for any input $x \in I_A$, $f(x)$ has a solution in S_B iff x has a solution in S_A.

In other words, if $A \leq_P B$ and we have a polynomial-time algorithm $\mathcal{A}_B$ to solve *B* then we have a polynomial-time algorithm to solve *A*, for any $x \in I_A$ computed in polynomial time $\mathcal{A}_B(f(x))$. Notice, a polynomial algorithm to solve *A* does not imply anything about the existence of a polynomial algorithm to solve *B.*

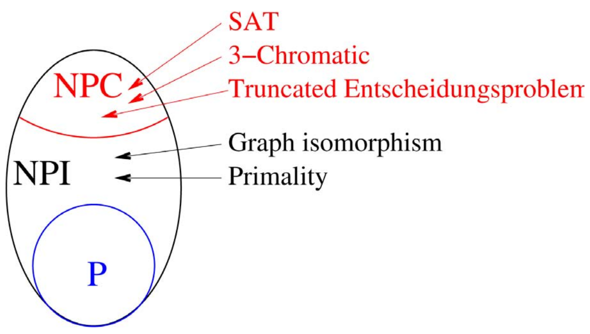
A search problem *B* is **NP**-complete if $B \in$ **NP** and for any **NP** problem *A*, we have $A \leq_P B$.

A useful property of reductions is that they are transitive. This property, together with the previous remark about reductions as problems solvers, tells us that the class **NP**-complete is the most difficult class of **NP** problems, in the sense that if one **NP**-complete problem is known to be in **P** then **P**=**NP.**

It is known that if **P≠NP,** there would be problems in **NP** that are neither in **P** nor **NP**-complete. These are in the class **NP**-intermediate (see Figure 2).

It is beyond the scope of this paper to show detailed reductions between **NP**-complete problems, which can be found in any standard algorithmic book, see for example Garey and Johnson, 1979; Moore and Mertens, 2011. But we would like to enumerate a few more examples of problems **NP**-complete and

Figure 2: Complexity classes inside **NP**



NP-intermediate. For a full taxonomy of **NP**-complete problems see Crescenzi and Kann (2012).

As already mentioned, SAT was the first problem known to be **NP-**complete; 3-SAT, the truncated Entscheidungsproblem, and chromatic number of a graph are also **NP**-complete problems. Factoring an integer as a product of primes is in **NP**-intermediate. Another significant problem in the class **NP**-intermediate is *graph isomorphism:* Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, find if there is a permutation $\pi: V_1 \rightarrow V_2$ such that $(u, v) \in E_1$ iff $(\pi(u), \pi(v)) \in E_2$.

Consider the *3-coloring of a graph problem:* "Given input graph $G = (V, E)$, decide if there is a valid coloring of *V* with exactly 3 colors" (see Figure 3). The problem is in **NP:** if $\mathcal{P}$ provides a 3-color witness χ, then $\mathcal{V}$ can verify in $O(|V|^2)$ whether χ is a valid coloring. It is known that this problem is **NP**-complete. Deciding whether a graph is *2-colorable* and, if so, finding the coloring is in **P** (Dasgupta, Papadimitriou and Vazirani, 2008).

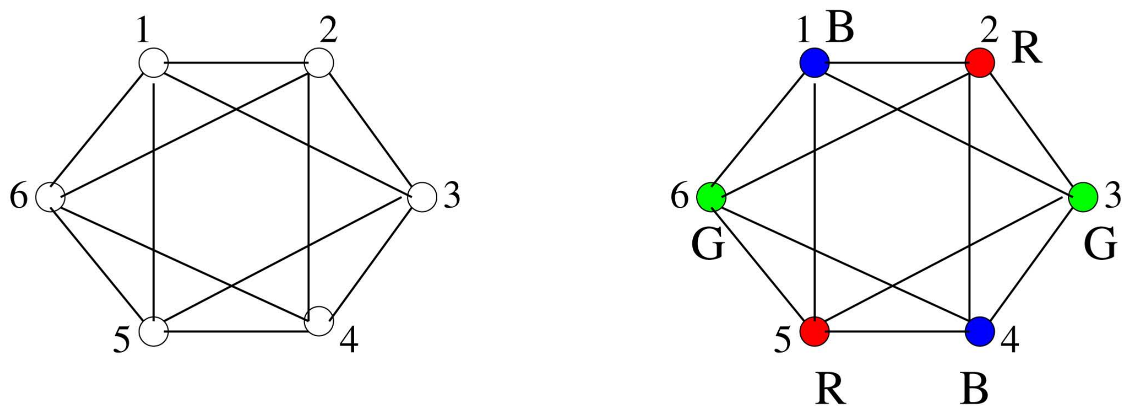
The following example taken from Moore and Mertens (2011, p. 152) shows that **NP**-completeness can appear in calculus problems. The *cosine integration problem:* "Given integers $x_1, \ldots, x_n$, decide if

$$\int_{-\pi}^{\pi} (\cos x_1\theta)(\cos x_2\theta) \cdots (\cos x_n\theta)\, d\theta \neq 0".$$

The class **NP**-complete contains many everyday practical problems. There is a series of problems dealing with minimizing delivery costs or time, which derive from the *Traveling Salesman Problem* (TSP). The search version of TSP has as input a complete graph $G = (V, E)$, with a weight $w(v_i, v_j) \geq 0$ on each edge (v_i, v_j), together with a value *c,* the goal is to find a tour visiting all vertices exactly once, such that

$$\sum_{(v_i,v_{i+1})\in\Pi(V)} w(v_i, v_{i+1}) \leq c.$$

Figure 3: 3-colorable G with a valid coloring



This problem is **NP**-complete. Another important

This problem is **NP**-complete. Another important set of practical problems derive from the *job scheduling problem.* In its most general setting, the problem can be formulated as follows: "Given n jobs J_1,... J_n, where each J_k has a processing time t_k, and given m identical machines M_1,..., M_m, then each job J_k must run on a machine M_i for t_k consecutive units of time, and during that time no other job can run on the same machine. The problem is to find an assignment of jobs to machines such that it minimizes the *makespan* of the schedule

$$T_{\min} = \max_{1\leq i\leq m} T_i,$$

where T_i denotes the time at which machine M_i completes its jobs". The search version of the problem is **NP**-complete for $m > 2$. For more information on the problem and its variations, see Brucker (2006).

As a final example, we would like to present evidence of the influence of the **P** versus **NP** problem in other disciplines, for example, economics. The *Efficient Market Hypothesis (EMH)* states that future prices cannot be predicted by analyzing prices from the past (Fama, 1965). In an enjoyable recent paper by the economist P. Maymin (2011), he proves that the *EMH* is false iff **P=NP**. For people with a little background in complexity theory, the paper is quite straightforward, still it presents a clear example of the spread of complexity theory to other fields.

4.3 Coping with NP-completeness

What can be done when having to deal with an **NP**-complete problem? For inputs of very small size n, a complexity of $O(2^n)$ or $O(n!)$ can be computed in reasonable computer time, but as we showed in Figure 1, the problem becomes intractable as n grows. Nevertheless, it turns out that for some **NP**-complete problems there are only a few "bad" inputs. The worst-case complexity assumes that there is a *devil adversary* that chooses the worst possible instance of the problem. In the early 1990's there was empirical evidence that for some problems such as graph coloring, satisfiability, integer partition, etc., there was a sharp jump from instances which were easily solved to instances that were easily shown not to have a solution, and just a few instances were difficult to solve. That phenomenon is denoted *phase transition* as it is similar to the phenomena studied by physics of sudden changes of states, between solid, liquid and gas. For instance, consider the 3-SAT problem. There is a well-know clever exhaustive search algorithm for SAT, the Davis-Putnam-Logemann-Loveland (DPLL), which was shown to solve many random instances of 3-SAT in polynomial time (Mitchell, Selman, and Levesque, 1992). To produce a random instance for 3-SAT with n boolean variables and $m = rn$ clauses, one has to choose uniformly at random each clause, with probability $1/\binom{n}{3}$ and then go over the variables in each clause and negate each with probability = 1/2. The *density* of one of these formulae is defined as $r = n/m$.

For example, the random 3-SAT formula

$$\phi_2 = (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee \bar{x_2} \vee x_4) \wedge (\bar{x_1} \vee \bar{x_2} \vee \bar{x_4}) \wedge (x_1 \vee \bar{x_3} \vee \bar{x_4}) \wedge (\bar{x_1} \vee \bar{x_2} \vee \bar{x_3}) \wedge (\bar{x_2} \vee x_3 \vee x_4) \wedge (\bar{x_1} \vee x_2 \vee \bar{x_3}) \wedge (\bar{x_2} \vee x_3 \vee \bar{x_4}) \wedge (x_2 \vee \bar{x_3} \vee \bar{x_4}) \wedge (\bar{x_2} \vee \bar{x_3} \vee \bar{x_4})$$

has density $r = 0.4$.

Mitchell, Selman, and Levesque (1992) first experimentally showed that the number of random 3-SAT formulae for which the DPLL algorithm took exponential time was small (see Figure 4a). Moreover, their experiments also showed that, with high probability, for densities $r < 4.2$ random 3-SAT formulae are satisfiable and for $r > 4.2$ random 3-SAT formulae are not satisfiable (see Figure 4b).

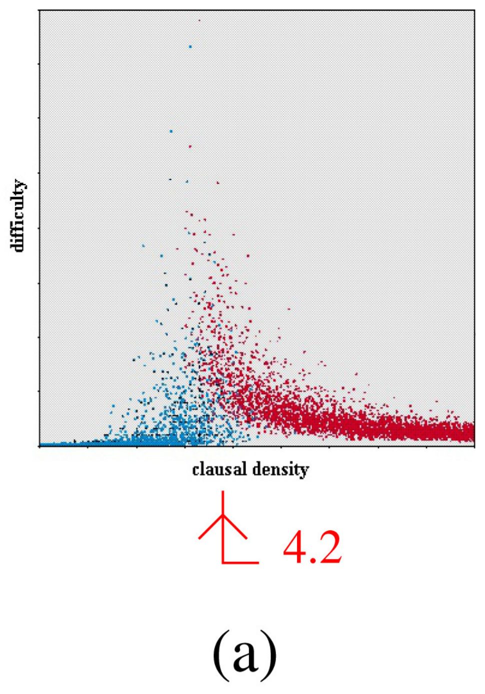
In 2002, using non-rigorous techniques from statistical physics (the replica method) on very large instances of 3-SAT, Mézard, Parisi and Zecchina (2002) and Mézard and Zecchina (2002) showed that the threshold for 3-SAT occurs at formulae with density $r_c = 4.27$, i.e., random 3-SAT formulae with density < 4.27 are satisfiable with high probability, and random 3-SAT formulae with density < 4.27 are satisfiable with high probability, and radom 3-SAT formulae with density > 4.27 are NOT satisfiable with high probability. Since then, there has been an effort to establish this sharp phase transition by rigorous analytical methods. So far for 3-SAT, the best lower bound is 3.52 (Hajiaghayi and Sorkin, 2003; Kaporis, Kirousis and Lalas, 2006) and the corresponding upper bound is 4.4907 (Díaz, Kirousis, Mitsche and Pérez, 2009). Closing the gap remains an open problem. See Chapter 14 in Moore and Mertens (2011).

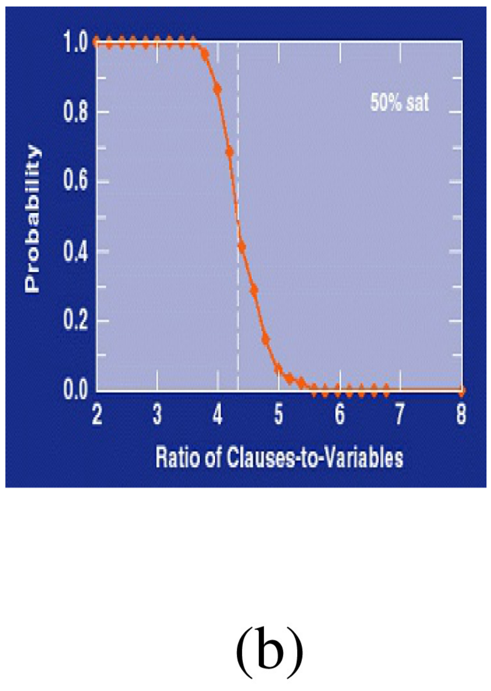
The fact that some **NP**-complete problems have a not too large number of bad instances indicates that sometimes *heuristics* can be used to achieve good results. Heuristics are procedures with no guarantees either in the running time or on the accuracy of the obtained solution, but for many hard problems, like for example, the layout of VLSI circuits, heuristics are the best practical solution (Wolf, 2011). Some modern textbooks on algorithms include a chapter on heuristics, clever backtracking techniques, like the DPLL algorithm we mentioned for solving SAT, simulated annealing or different versions of local search. For a general textbook on heuristics see Michalewicz and Fogel (1998).

Another practical alternative is using approximation algorithms. An algorithm is said to *r-approximate* an optimization problem if, on every input, the algorithm finds a solution whose cost is $\leq 1/r$ if the problem asks for the minimum value, or $\geq 1/r$ if the problem asks for a maximum value.

Almost since the beginning of the development of complexity theory, there was a parallel effort to develop approximation algorithms for **NP**-complete problems. Although the first such algorithm is due to Graham in 1966, who developed it to approximate a version of scheduling, the seminal paper for approximation theory by Johnson

Figure 4: Experiments with random 3-SAT formulae, from Mitchell, Selman, and Levesque (1992). In (a) each dot represents the time DPLL takes on a random instance of 3-SAT. Light dots represent satisfiable instances and dark dots represent non-satisfiable instances. (b) The phase transition satisfiable to non-satisfiable for a random 3-SAT formula occurs at density 4.27.





(1974) came very little after Cook, Levin and Karp's papers. Since then, the theory of approximation and of inapproximability has become a very fruitful field in theoretical computer science, with good books covering the topic, for example Williamson and Smoys, 2010.

Parallelism is another powerful tool to speed up computation by a constant factor. Notice that anything that can be done with 10^{10} processors in T steps, can be done with one processor in $10^{10}T$ steps. But unless **P**=**NP**, the difference between **P** and **NP**-complete problems is an exponential cost of computation, therefore parallelism would not be the tool to solve **NP**-complete problems in polynomial time.

In the same way, a result by Impagliazzo and Widgerson (1997) implies that, unless **P**=**NP**, randomization would not help to solve **NP**-complete problems. In fact, there is a stronger conjecture: randomization mainly helps to improve the time complexity of problems in **P**.

At the present time, quantum computers are not an existing reality, although *D-Wave Systems, Inc.* has managed to build a 128-qubit processor (not general purpose). At the theory level, we know quantum computation could solve **NP**-intermediate problems, for example factorization of an integer (Shor, 1997), but it is generally agreed that, unless **P**=**NP**, quantum computation would not help with **NP**-complete problems (Bernstein and Vazirani, 1997). See Aaronson (2008) for an extensive discussion on the limitations of quantum computers.

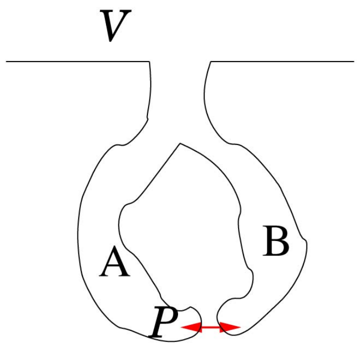
5. BEYOND TURING: INTERACTIVE PROOFS AND ZERO-KNOWLEDGE PROOFS

A fascinating line of research in theoretical computer science started in the mid 1980's, which led to fruitful results, namely *Interactive Proofs.* In this section, we aim to give a brief intuitive introduction to the topic, in particular to *Zero-Knowledge Proofs* (ZKP) and *Probabilistically Checkable Proofs (PCP).* For the interested reader, we recommend Chapter 11 in Moore and Mertens (2011) and Chapters 8, 9 and 11 in Arora and Barak (2009). The topic has spanned 26 years, with numerous papers and fruitful applications in various fields, such as cryptography and approximation algorithms, among others.

The basic idea is how one researcher *(the prover $\mathcal{P}$)* can convince another researcher *(the verifier $\mathcal{V}$)*[6] that he has a correct proof to a difficult theorem by only showing to $\mathcal{V}$ a few random bits of the proof, in such a way that at the end $\mathcal{V}$ is convinced that the proof is correct without having any insight into how it works.

To grasp the concept of zero-knowledge proof, let us start with the very simple illustrative example from Quisquater, Quisquater, Quisquater, Quisquater, Guillou, Guillou, Guillou, Guillou, Guillou, Guillou, and Berson (1989)[7]. There is a cave with two paths *A* and *B,* which are connected at their ends by a secret passage (see Figure 5). To cross that passage one must know the magic words. In our case, $\mathcal{P}$ wants to convince $\mathcal{V}$ that he knows the magic words without telling them to $\mathcal{V}$. They agree on the following protocol: $\mathcal{V}$ will remain outside of the cave, so he cannot see which path $\mathcal{P}$ takes. When $\mathcal{P}$ arrives at the end of the cave, $\mathcal{V}$ tells $\mathcal{P}$ which path to return along, and $\mathcal{V}$ enters the cave to make sure $\mathcal{P}$ is returning for the path he indicated. The probability that $\mathcal{P}$ took the path $\mathcal{V}$ asks him to return is 1/2, therefore if the experiment is repeated a sufficiently large number of times, and each time $\mathcal{P}$ returns by the correct path, with probability 1 $\mathcal{P}$ must know the magic words to cross the cave, and $\mathcal{V}$ is convinced that $\mathcal{P}$ knows them.

Figure 5: Ali Baba's magic cave.



ZKP are a particular case of the most general *Interactive Proofs Systems,* introduced concurrently in Babai (1985) and Goldwasser, Micali and Rackoff (1989)[8]. Technically the word *proof* refers to a randomized *interactive protocol* between $\mathcal{P}$ and $\mathcal{V}$, where $\mathcal{P}$ has unlimited computational capabilities and tries to convince $\mathcal{V}$ of the truth of a certain statement. Loosely speaking, the two characteristics that an interactive protocol must have to be an interactive proof is that an honest $\mathcal{V}$ should always be convinced by an honest $\mathcal{P}$, but a cheater $\mathcal{P}$ should have a very small probability of convincing an honest $\mathcal{V}$ that a false statement is true. The interactive proof system is zero-knowledge if $\mathcal{V}$ is not going to learn anything from the interaction with $\mathcal{P}$. In the previous example, $\mathcal{V}$ becomes convinced that $\mathcal{P}$ knows the magic words to cross between the two paths.

Let us describe a more interesting example. Consider a given graph $G = (V, E)$, where $|V| = n$ and $|E| = m$. As we saw before, to decide whether G has chromatic number 3 is an **NP**-complete problem, therefore in general it is not a feasible problem to solve for large values of n. In this setting, $\mathcal{P}$ wants to convince $\mathcal{V}$ that he has a valid 3-coloring of G, without revealing the coloring. This example is from Goldreich, Micali and Wigderson (1991). At each iteration of the protocol, $\mathcal{V}$ only has access to the colorings at the ends of a single edge he chooses at the iteration. The protocol is the following: First $\mathcal{P}$ selects a valid 3-coloring (for every $(u, v) \in E$, u and v must have different colors) and generates all 3! = 6 permutations of valid 3-colorings for G; let C be the set of all 6 valid colorings. For n^3 iterations, at each iteration i, $\mathcal{P}$ chooses with probability 1/6 a new coloring $c_i \in C$, $\mathcal{V}$ selects an edge and verifies that the edge is correctly colored. Assume that G has a valid 3-coloring (see for example Figure 3). Notice that if $\mathcal{V}$ chooses the same edge (u, v) at two different iterations, the colors assigned to each vertex may be different, as at each iteration $\mathcal{P}$ chooses a new random coloring, therefore $\mathcal{V}$ will not be able to learn a valid coloring for the whole G. On the other hand, if G is not 3-colorable, at each round, at least one edge will have the same color for both vertices, therefore the probability that $\mathcal{V}$ will discover a wrong edge is at least $1/m$ per round. As $m \leq n^2$, after n^3 iterations with probability tending to 1, $\mathcal{V}$ will discover that G is not 3-colorable.

Technically the way $\mathcal{P}$ shows the colors to $\mathcal{V}$ is a bit more complicated, using a *one-way-function.* One-way-functions are functions that can be easily computed but are hard (exponential time) to invert. A trivial example of one-way-function is *integer multiplication:* it is easy to multiply $m = a_1 \times a_2 \times \ldots \times a_n$, however as we already mentioned, there is no known polynomial-time algorithm for factorizing m. Another more interesting example of a one-way-function is the *discrete logarithm:* "Given n-bit integers x, y, z, find whether there exists an integer w such that $y = x^w \bmod z$". Given x, w and z, it is easy to compute $y = f(x, w, z) = x^w \bmod z$, but it is conjectured that finding $f^{-1}(y)$ takes exponential time. One-way-functions are in standard use in cryptography.

By applying reductions to the 3-colorability problem, it was shown in Goldreich, Micali and Wigderson (1991) that under the assumption of the existence of one-way-functions, every problem in class **NP** has a zero-knowledge proof. In fact, if we denote **IP** the class of problems having an interactive protocol, it is known that **IP**, as a complexity class, contains far more difficult problems than **NP** (under the hypothesis **P≠NP**) (Shamir, 1992). Computing the permanent of a matrix is a problem in the class **IP**, which means that even under the hypothesis **P=NP**, it would remain a hard problem.

The culmination of interactive proof systems research was one of the most beautiful and deep theorems in computer science, the *PCP-theorem.* Although the Gödel prize 2001 was shared by Arora and Safra (1998), Arora, Lund, Motwani, Sudan and Szegedy (1998) and Feigue, Goldwasser, Lovasz, Safra, and Szegedy (1996) for their contribution to *Probabilistically Checkable Proofs* and the PCP-theorem, many of the techniques and ideas are due to a much larger number of researchers (see for example Johnson (1992) for an extensive historical account, and Chapter 16 in Williamson and Smoys (2010) for further recent work using the PCP-theorem to obtain inapproximability results).

The rough idea of probabilistically checkable proof systems is: "Given a conventional mathematical proof in which a mistake could be hidden in any equation, transform the proof in such a way that the mistake is spread almost everywhere". This kind of proof is denoted a *holographic* proof. A PCP system for an **NP** problem encodes the *witness* to the problem in a way such that $\mathcal{V}$ can verify probabilistically the witness by looking only to a few of its bits, so that if it is true $\mathcal{V}$ accepts with probability 1, and if it is false $\mathcal{V}$ accepts with probability < 1/2.

The *PCP-theorem* states that holographic proofs exist for problems in **NP,** i.e., that any problem in **NP** has a polynomial length probabilistically checkable proof, where $\mathcal{V}$ flips $O(\log n)$ random coins and need to look only at $O(1)$ bits of the proof.

6. CONCLUSIONS

Contrary to Hilbert's Entscheidungsproblem, it remains an open problem to decide whether the truncated Entscheidungsproblem is feasible, i.e., it remains open to decide whether **P=NP**. It follows from the arguments in the present paper that a positive answer to that question may answer all seven remaining open millennium problems.

At least there are 54 existing bogus proofs of the **P=NP** question. Of them, 26 "proving" the equality, 24 "proving" the strict inclusion, and 3 "proving" that the **P=NP** question is itself undecidable. For further details see Woeginger. Most scientists working in complexity theory believe that **P≠NP,** but there are some top scientists in the field that disagree with the majority, see for instance Lipton's blog.

The aim of this manuscript was not to review the complexity field or the status and future of the **P**=**NP** question. As we pointed out, there are some outstanding textbooks dealing with all the past, present and future attempts and results in complexity. Our only incursion into the areas of modern complexity has been the topic of interactive proofs, and this is because we think that it is a natural continuation to the truncated Entscheidungsproblem, in the sense that PCP basically tells us how to convince our colleagues that we have a correct proof without giving away any real details. Moreover, it turns out that PCP is a strong characterization of **NP** problems. Could we one day even have a practical holographic way to check proofs?

Complexity theory has studied different models of computation (Turing machine with bounded number of steps, Boolean circuits, quantum algorithms, randomized algorithms), the complexity of measuring different parameters (space, number of iterations, depth of circuits), and several measures of complexity (worst case, average, smoothed). All this work has created a whole cosmos of about 500 complexity classes (Aaronson, Kuperberg and Granade). For most of them, strict relations are not known and, at the end, the main issue boils down to the basic intuition by Kurt Gödel in 1954.

We tried to convey the idea that this is one of the deepest questions of today's science, affecting not only computer science, mathematics and physics, but also biology, sociology, economics and most human activities. We see this broad coverage of this question as a direct consequence of Turing's way of looking through *the algorithmic lens* to problems in different disciplines, from cryptography and physics to biology. The spread of modern technologies is accelerating the need for an *algorithmic view* of today's social, economical, cultural, and political interactions, leading directly to the question **P=NP**? For an excellent survey of the role of the algorithmic view into the activities within the modern world, we recommend the book by Easley and Kleinberg (Easley and Kleinberg, 2010).

NOTES

1 In parallel to Turing, A. Church also gave a negative answer to the Entscheidungsproblem using a logic formalism, λ-calculus.

2 The complexity is expressed in asymptotic notation, i.e., for very large values of the input. The notation $T(n) = O(f(n))$ means that $\lim_{n\to\infty} T(n)/f(n) = c$, where c is a constant.

3 It may happen that there is no feasible solution for input x, then $k(x, s)$ does not exist.

4 The correct word is recognized, as the problems are posed as recognition problems, i.e. determining whether a word belongs to a language over a finite alphabet.

5 A positive answer will render all transactions done using RSA insecure.

6 In a large part of the scientific bibliography, the prover and the verifier are respectively named Merlin and Arthur.

7 The authors frame their explanation in the arabic tale "Ali Baba and the forty thieves" from the classic "One thousand and one nights".

8 The conference version of Goldwasser, Micali and Rackoff (1989) appeared at STOC-85.

REFERENCES

Aaronson, S. (2008). "The limits of quantum computers". *Scientific American,* 289, pp. 62-69.

Aaronson, S.; Kuperberg, G. and Granade, C. *Complexity Zoo*. http://qwiki.stanford.edu/index.php/Complexity_Zoo.

Agrawal, M.; Kayal, N. and Saxena, N. (2002). "Primes is in P". *Annals of Mathematics,* 2, pp. 781-793.

Arora, S. and Barak, B. (2009). *Computational Complexity: A Modern Approach.* Cambridge University Press.

Arora, S.; Lund, C.; Motwani, R.; Sudan, M. and Szegedy, M. (1998). "Proof verification and the hardness of approximation problems". *Journal of the ACM,* 45 (3), pp. 501-555.

Arora, S. and Safra, S. (1998). "Probabilistic checking of proofs: A new characterization of NP". *Journal of the ACM,* 45 (1), pp. 70-122.

Babai, L. (1985). "Trading group theory for randomness". In *Proc. 17th. ACM Symposium on the Theory of Computing,* pages 421-429.

Bernstein, E. and Vazirani, U. V. (1997). "Quantum complexity theory". *SIAM Journal Computing,* 26 (5), pp. 1411-1473.

Brucker, P. (2006). *Scheduling Algorithms.* Springer, fifth edition.

Cook, S. (1971). "The complexity of theorem-proving procedures". In *3rd. ACM Symposium on the Theory of Computing,* pages 151-158.

Cormen, T. H.**;** Leiserson, C.; Rivest, R. and Stein, C. (2001). *Introduction to Algorithms.* The MIT Press, 3 edition.

Crescenzi, P. and Kann, V. (2012). A compendium of NP optimization problems.

Dasgupta, S.; Papadimitriou, C. and Vazirani, U. (2008). *Algorithms.* McGraw-Hill.

Davis, M. (2000). *The universal computer: the road from Leibniz to Turing.* Norton.

Díaz, J.; Kirousis, L.; Mitsche, D. and Pérez, X. (2009). "On the satisfiability threshold of formulae with three literals per clause". *Theoretical Computer Science,* 410, pp. 2920-2934.

Doxiadis, A.; Papadimitriou, C.; Papadatos, A. and di Donna, A. (2009). *LOGICOMIX: an epic search for truth.* Bloomsbury.

Easley, D. and Kleinberg, J. (2010). *Networks, Crowds and Markets. Reasoning about a highly connected world.* Cambridge University Press.

Edmonds, J. (1965). "Paths, trees, and flowers". *Canad. J. Math.,* 17, pp. 449-467.

Fama, E. (1965). "The behavior of stock-market prices". *The Journal of Business,* 38 (1), pp. 34-105.

Feigue, U.; Goldwasser, S.; Lovasz, L.; Safra, S. and Szegedy, M. (1996). "Interactive proofs and the hardness of approximating cliques". *Journal of the ACM,* 43 (2), pp. 268-292.

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

Glynn, D. G. (2010). "The permanent of a square matrix". *European J. of Combinatorics,* 31 (7), pp. 1887-1891.

Goldreich, O.; Micali, S. and Wigderson, A. (1991). "Proofs that yield nothing but their validity or all languages in NP have Zero-Knowledge Proof Systems". *Journal of the ACM,* 38 (1), pp. 691-729.

Goldwasser, S.; Micali, S. and Rackoff, C. (1989). "The knowledge complexity of interactive proof systems". *SIAM J. Computing,* 18 (1), pp. 186-208.

Graham, R. (1966). "Bounds for certain multiprocessing anomalies". *Bell System Technology Journal,* 45, pp. 1563-1581.

Hajiaghayi, M. T. and Sorkin, G. (2003). The satisfiability threshold of random 3-SAT is at least 3.52. Technical report, IBM Research Report.

Hartmanis, J. and Steam, R. (1965). "On the computational complexity of algorithms". *Transactions of the American Mathematical Society,* 117, pp. 285-306.

Impagliazzo, R. and Wigderson, A. (1997). "*P = BPP* if *E* requires exponential circuits: Derandomizing the XOR lemma". In *Proceedings of tTwenty-Ninth Annual ACM Symposium on the Theory of Computing,* pages 220-229.

Johnson, D. J. (1974). Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences,* 9, 256-278.

Johnson, D. S. (1992). "The NP-completeness column. The tale of the second prover". *Journal of Algorithms,* 13 (3), pp. 502-524.

Kaporis, A. C.; Kirousis, L. and Lalas, E. G. (2006). "The probabilistic analysis of a greedy satisfiability algorithm". *Random Struct. Algorithms,* 28 (4), pp. 444-480.

Karp, R. M. (1972). "Reducibility among combinatorial problems". In R. E. Miller and J. W. Thatcher (eds.), *Complexity of Computer Computations,* pp. 85-104. NY: Plenum Press.

Levin, L. (1973). "Universal sequential search problems". *Probl. Peredachi Inf.,* 9, pp. 115-116.

Lipton, R. Godel's lost letter and P = NP. http://rjlipton.wordpress.com.

Lipton, R. (2010). *The P=NP Question and Godel's Lost Letter.* Springer.

Maymin, P. (2011). "Markets are efficient if and only if P=NP". *Algorithmic Finance,* 1, pp. 1-11.

Mezard, M.; Parisi, G. and Zecchina, R. (2002). "Analytic and algorithmic solution of random satisfiability problems". *Science,* 297 (812).

Mezard, M. and Zecchina, R. (2002). "The random k-satisfiability problem: from an analytic solution to an efficient algorithm". *Physics Review,* E 66-056126.

Michalewicz, Z. and Fogel, D. (1998). *How to solve it: Modern Heuristics.* Springer.

Mitchell, D.; Selman, B. and Levesque, H. (1992). "Hard and easy distributions of sat problems". In *Proceedings of the 10th. National Conference on Artificial Intelligence (AAAI),* pp. 459-465.

Moore, C. and Mertens, S. (2011). *The Nature of Computation.* Oxford University Press.

Nissan, N. (2004). John Nash's letter to the NSA. http: //agtb.wordpress.com/2012/02/17/john-nashs-letter-to-the-nsa/, February 17.

Pratt, V. R. (1975). "Every prime has a succinct certificate". *SIAM J. Comput.,* 4 (3), pp. 214-220.

Quisquater, J.-J.; Quisquater, M.; Quisquater, M.; Quisquater, M.; Guillou, L. C.; Guillou, M. A.; Guillou, G.; Guillou, A.; Guillou, G.; Guillou, S. and Berson, T. A. (1989). "How to explain zero-knowledge protocols to your children". In G. Brassard, editor, *CRYPTO-89,* volume 435 of *Lecture Notes in Computer Science,* pp. 628-631. Springer.

Shamir, A. (1992). "IP = PSPACE". *Journal of the ACM,* 39 (4), pp. 869-877.

Shor, P. W. (1997). "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer". *SIAM J. Comput.,* 26 (5), pp. 1484-1509.

Singh, S. (2000). *The Code Book.* Anchor Books.

Trakhtenbrot, B. (1984). "A survey of russian approaches to perebor (brute-force searches) algorithms". *Annals of the History of Computing,* 6 (4), pp. 384-400.

Turing, A. M. (1939). Systems of logic based on ordinals. *Proceedings of the London Mathematical Society-2,* 45, pp. 161-228.

Valiant, L. G. (1979). "The complexity of enumeration and reliability problems". *SIAM J on Computing,* 8, pp. 410-421.

Williamson, D. and Smoys, D. (2010). *The Design of Approximation Algorithms.* Cambridge University Press.

Woeginger, G. The P vs. NP page. http://www.win.tue.nl/ gwoegi/P-versus-NP.htm.

Wolf, W. (2011). *Modern VLSI Design.* Prentice-Hall, fourth edition.