# Knowledge, Creativity and $P$ versus $NP$

Avi Wigderson

April 11, 2009

**Abstract**

The human quest for efficiency is ancient and universal. Computational Complexity Theory is the mathematical study of the efficiency requirements of computational problems. In this note we attempt to convey the deep implications and connections of the results and goals of computational complexity, to the understanding of the most basic and general questions in science and technology.

In particular, we will explain the $P$ versus $NP$ question of computer science, and explain the consequences of its possible resolution, $P = NP$ or $P \neq NP$, to the power and security of computing, the human quest for knowledge, and beyond. The connection rests on formalizing the role of creativity in the discovery process.

The seemingly abstract, philosophical question: *Can creativity be automated?* in its concrete, mathematical form: *Does* $P = NP$*?*, emerges as a central challenge of science. And the basic notions of *space* and *time*, studied as resources of efficient computation, emerge as key objects of study to solving this mystery, just like their physical counterparts hold the key to understanding the laws of nature.

This article is prepared for non-specialists, so we attempt to be as non-technical as possible. Thus we focus on the intuitive, high level meaning of central definitions of computational notions. Necessarily, we will be sweeping many technical issues under the rug, and be content that the general picture painted here remains valid when these are carefully examined. Formal, mathematical definitions, as well as better historical account and more references to original works, can be found in any textbook, e.g. see [14, 17]. We further recommend the surveys [19, 3, 18, 11] for different perspectives on the $P$ versus $NP$ question, its history and importance.

# 1 Introduction - humanity's quest for knowledge

Consider the basic notions of *time, space, energy, mass*, and two questions regarding them.

**Question 1**

A basic problem faced by physicists is *what are they?* Attempting to understand the universe, come up with a theory that explains (better than previous theories) their nature, structure, interaction, origin,....

**Question 2**

A basic problem faced by the high-tech industry, is *how to minimize them?*. For example, design a cell phone with video and Internet capacities, that is *faster, smaller, cooler, lighter* (than previous cell phones).

There are many obvious differences between these questions. In particular, the roles played by the basic notions of time, space, energy and mass in each are entirely different – in the first they are "neutral" objects of study, while in the second they are scarce resources, whose use we are trying to control.

But are these two questions related? There are of course some obvious relations. For instance, humanity seems obsessed with pursuing both. While the reasons may differ, in both cases there is willingness to invest huge efforts to answer, or better understand them.

In this article we exhibit a far deeper connection between these questions. Solving Q2 and questions like it may well be relevant (or even sufficient) for solving Q1. Furthermore, this connection extends to most questions in mathematics, science and technology pursued in our quest for knowledge. We will argue that the $P$ versus $NP$ question – the central question of computer science, holds the key to our ability to finding answers to questions we raise! We shall see that if $P = NP$ humanity can expect to answer most questions it can pose. And we'll also see how (stronger versions of) the possibility $P \neq NP$, present enormous benefits to humanity as well, many of which we are experiencing daily.

Thus the answer to an abstract mathematical question, "Does $P = NP$?", carries a fundamental insight into the nature of our world, and what can we hope to know about it. As such, it stands with the grand challenges, like uncovering the laws of nature, explaining the workings of the human brain and body, etc. At the same time $P = NP$? addresses our ability to meet these challenges, and in Gödelian style, our ability to answer if $P = NP$?.

We start by explaining the fundamental connection between Q1 and Q2 above. As these represent respectively science and technology, let us add

another question Q3 to represent mathematics. Q3 again seems as different from Q1 and Q2 as they are from each other.

**Question 3**

Fermat's last "theorem"[1]: Can the equation $X^n + Y^n = Z^n$ be solved with positive integers $X, Y, Z$ for any $n > 2$ ?

# 2 The connection: efficient verifiability

The main thesis of this section is:

*For many interesting and important questions we humans raise and seriously study, we expect to be able to* **efficiently recognize** *a solution when given or found.*

Let us examine this thesis critically for the 3 questions above, as well as for the fields of study they represent. We hope to make a convincing case for this thesis, at least in this informal form. We shall later try to formalize the word *recognize* and stress that *efficiency* is essential for its meaning. While this may seem too "mechanical" or "computational" in some cases, we hope to at least make the case that probing to what extent such recognition can be so formalized, is an extremely important direction of study.

## 2.1 Mathematics

For Q3, indeed for all questions of mathematics, the thesis is obviously true. After all, mathematics is characterized, and set apart from all other fields of study, by the notion of *proof*. The meaning of resolving a mathematical question **is** coming up with a formal proof for the associated claim. While it has taken over three centuries and monumental work to understand the relevant underlying mathematical structures, once Andrew Wiles wrote his (200 page) proof it became possible for the mathematical community to verify its correctness in a couple of years. Moreover, for most known mathematical theorems, proofs can verified by competent mathematicians in a matter of days. We stress that this is not just verification – it is an *efficient* verification. This ability allows the mathematical community to reach consensus on truth, and constantly enlarge its knowledge base.

To make the efficiency point even stronger, we note that Wiles wrote his proof in the common style of math papers, with lots of (standard) abbreviations and references to past work. But had Wiles written his proof completely formally (namely, in the syntax of the appropriate logical system),

[1] Yes, we know it was solved

while it may have taken 20,000 pages rather than 200, a simple computer program would have been able to verify the correctness of the proof in 1 second! That's fantastically efficient. And moreover, work is underway to allow math journal type proofs to be automatically converted to their logical syntax format (again in 1 second), releasing Wiles from the chore above and letting him focus on his next challenge. The recent survey [20] describes such projects, including formalizing and verifying the extremely complex (and computer-aided) proofs of the four-color theorem and the Kepler conjecture.

### 2.2 Science

Science studies the real, physical world, rather than the pure abstract one of mathematics. Here there is no absolute truth, as the best understanding we have of any phenomena is only as good as the observations relating to it that we were able (so far) to make. Science progresses by a continuous interaction between theorists and experimentalists. The first group suggests theories to explain the available data. The second conducts experiments and observations to enlarge the available data and test the theories.

In both activities, results are announced and papers are written only after their authors feel that they can convince their community that something new was discovered. Again, the efficient recognition of novel ideas and new phenomena is key to the progress in science. This holds not only in famous cases that everyone learns about in school; theories like Newton's mechanics, Maxwell's electromagnetism, Einstein's relativity, etc,. and experimental discoveries like radioactive decay and the cosmic microwave background radiation. It happens hundreds of times a day around the world - scientists realize they are on to something, and can effectively convey it to their colleagues. And this of course holds not only for physics, but for all sciences, including some of the social sciences.

Unlike mathematics, the efficient verification process clearly exists, but is not fully specified (or even completely conscious), and indeed may differ between areas of scientific activity. However, we believe that it is generally possible for scientists to *explicitly* specify reasonably formally the type of models they seek when attempting to explain some phenomena. Very generally, these are concise models which are typically algorithmic; be it Markov models, system of differential equations, etc. such (predictive) models often describe the evolution of some system from one state to the next, in a manner that can be efficiently tested to be consistent with given data.

This point, about the ability to convert an *abstract* notion of recognition

of discovery, into a *concrete*, efficient program to do so, is essential to this article. Sometimes scientists are forced to be so explicit! Let us give but one example to demonstrate the efforts of scientists to fully specify a procedure to recognize *new* knowledge. While perhaps atypical, it hopefully shows that "if there's a will, there's a way", and that motivation can generate similar examples in many other situations.

Let's go back to Q1, and to what is perhaps the largest (the budget is several billion dollars) experiment, designed to further our understanding of time, space, energy, mass and more generally the physical laws of nature. We refer to the LHC (Large Hardon Collider) at CERN, Geneva, which should be operational in about one year. The outcomes of its experiments are awaited eagerly by numerous physicists. If these do confirm (or refute) such theories like *supersymmetry*, or the existence of the elusive *Higgs particle* (responsible for mass in the current prevailing theory, *the standard model*), this excitement will be shared with the public at large.

But how would they know? The LHC bombards protons against each other at enormous energies, and detectors of all kinds attempt to record the debris of these collisions. There are billions of collisions per second (and the LHC will operate for several years), so the total data detected is *much, much* larger than what can be kept. Moreover, only a few of these collisions (less than 1 in a million) provide *new* information relevant to the searches above (most information was already picked up by previous, less sensitive colliders, and helped build up the current theories). So ultrafast on-line computer decisions have to be made to decide which few information bits to keep! The people who wrote these programs have designed and implemented an efficient recognition device for new knowledge! Needless to say, the programs that would search and analyze the kept data (20-mile high stack of cd's of it), would have to be designed to efficiently find something new. Ultimately, we would like programs that would analyze the data and suggest new models and theories explaining it directly.

But clearly, such huge investment of resources would never take place if we were not convinced that new phenomena, if observed, could be efficiently recognized.

### 2.3 Engineering

Let us proceed to Q2, and the design of a new cell phone. What is the efficient verification of its properties? This is something we do on a daily basis as consumers. When the new design hits the stores, we go there, experiment with it to see that we like the Internet speed, the resolution of

the display, its weight, battery size, and of course its price, all of which can be precisely defined and measured[2].

But this is the usual story with engineering. The proof of the pudding, as they say, is in the eating. When an engineering question is solved, an implementation usually follows. Be it a cell phone, a bridge, a spaceship, a new cancer drug, a molecular motor, or the good old fashioned wheel; once designed, its discoverers (and us) can test to see that it satisfies all the design constraints. And these tests had better be efficient, or we will never know when we are done designing...

True, this field is far more diverse than the other two. Verification of correctness of mathematical theorems, or of the novelty and consistency of scientific theories, are academic in nature and are conducted mainly by academicians. But while engineering crucially depends on math and science to achieve its goals, its tasks are spread throughout all aspects of life. The nature of engineering constraints is very diverse, and the ingenuity found in the designs is often hard to precisely quantify.

Nevertheless, each particular design task, like Q2, does specify a set of constraints, which are usually quite well defined and efficiently testable, or no one would take it on. Typical commercial producers certainly specify for themselves a series of tests and evaluations which a potential product has to pass before being manufactured.

### 2.4 Economics and Politics

Finding the best design under a complex set of constraints is not limited to engineering. Replacing the word "design" by "action", we realize immediately that people, companies, and governments are continuously challenged by problems of exactly this sort: setting the prices and profit margins of products in the presence of competition, choosing the right interest rates to balance growth and unemployment, making tactical and strategic decisions in war, making the next move in a chess game, etc. The interactive nature of these problems makes it harder to compare different actions. However, given that people, companies and governments do make these decisions daily means that they each have their own (efficient!) criteria for preferring some actions to others. Are they making the best choices given these criteria?

[2]We may also consider some subjective things like color, style, etc. which are harder to quantify generally, but perhaps can be quantified personally.

### 2.5 Humanities and the Arts

Pushing our luck a bit, we'd like to speculate that, perhaps in all creative ventures we can imagine efficient verification taking place. While no specific criteria are necessarily set in advance and no specific test is necessarily applied to an outcome, we can speculate about the process allowing a philosopher or an artist to know that they have successfully completed something. Similarly, our satisfaction when hearing a piano sonata or reading a novel, can be explained perhaps by the successful completion of some (highly individual, but certainly efficient) process of evaluation.

## 3 The classes $P$ and $NP$

$P$ and $NP$ are collections of problems. Their informal definition is given below.

$P$ is the collection of all tasks for which a solution can be efficiently *computed*.

$NP$ is the collection of all tasks for which a solution, *when given*, can be efficiently *verified*.

In the two subsections below we exemplify problems in each class.

### 3.1 The class $NP$

Let us start with $NP$, for which we have already accumulated examples. Again, $NP$ is the collection of all those tasks for which a solution, *when given*, can be efficiently *verified*. The definition and importance of this class evolved from the seminal papers of Cook [2], Levin [12] and Karp [13] in the early 1970s.

For computational tasks, this classification has transformed computer science. For numerous practical problems, especially optimization problems arising in the industry, verification of good solutions is precisely defined, so all these problems formally lie in $NP$. It became clear that numerous man-years have and are being invested in trying (and often failing) to find efficient general solutions to $NP$ problems[3]. The remarkable unity of purpose of all these computer programmers and engineers working on diverse, seemingly unrelated problems, will be explained in section on *universality* 5.

As we have tried to argue in the previous section, $NP$ actually captures far more than computational tasks. After appropriately formalizing the

[3]Naturally, in industry failure is not well accepted, and often requirements are relaxed and approximate or heuristic solutions are used

recognition process, all mathematical tasks, many engineering tasks, and central scientific tasks as well fall into $NP$. We stress again our belief that the seeming dichotomy between *abstract* and *concrete* recognition can be bridged for great many of these problems. While the abstract recognition of discovery is sometimes elusive, it is often simply because no motivation existed for formalizing it (and turning it to an algorithm as required by the formal definition of $NP$). We feel it can be done for a vast number of the problems above. Accepting this, $NP$ constitutes a huge portion of the new knowledge, understanding and technology we seek.

Beyond grand challenges, everyday problems of finding a short route to our destination, solving puzzles, trying to fit all suitcases to the trunk of our car (we shall return to this one), scheduling our day under the constraints of others, etc., are also all in $NP$. These are faced on a small scale by each of us, and on a larger scale by essentially every private company and public body, trying to optimize the use of various resources in the presence of various constraints. And again here, the recognition of good solutions often is, or can be, fully specified and simply verified.

Which of this vast array of $NP$ problems can we ever hope to efficiently solve? Can we solve them *all*?

Sure, *some* problems in $NP$ we can solve. Adding 1+1, or adding any two integers, is a problem in $NP$ (since we have a quick way of verifying an answer when given). But even if the answer wasn't given, we could quickly find the answer ourselves. This leads us to another important class of problems, called $P$, of problems we can *solve* efficiently.

### 3.2 The class $P$

$P$ is the collection of all those tasks for which a solution can be efficiently *computed*. The definition and importance of this class evolved from the seminal works of Cobham [1], Rabin [15] and Edmonds [5] in the 1960's.

As mentioned, integer addition is in $P$. Indeed, a fast method for "long addition" is taught in first grade, and enables us to add *any* two integers in a number of steps which is proportional to their length. That's as efficient as you can get – just reading the data takes that long!

Such a "method" is called an *algorithm* – the fundamental recipe for solving a given task (like addition) on every input data. An algorithm is the intellectual heart of any computer program solving that task.

We learn and discover many efficient algorithms in school and in life, showing that the relevant tasks are in $P$. Algorithms for other arithmetic operations like "long multiplication" are also taught early on, and are also

efficient (but not as quick as addition). Sorting a deck of cards after a game can also be efficiently done, with any number of cards. Searching for a name in a 1000 page phone book (or any sorted list) is never done sequentially, going from page 1 to page 2 to page 3 until we find the name. This might actually take a 1000 steps. We invariably use the much faster "binary search" algorithm: we look at page 500, and determine if the name we seek is before it or after it. Then we go to the middle of the relevant half, and do the same. Repeating this will converge to the location of our name in merely 10 steps.

What we can quickly do by hand with an efficient algorithm on small data, the computer does extremely quickly even on huge data. Essentially anything you see implemented, is solving a problem in $P$, backed by the necessary efficient algorithm. If your desktop does searches, spell checks, Excel pie charts, cool graphic and computer games, it is only due to efficient algorithms solving the underlying computational problems. If, in 1 second, your e-mail is sent to the other side of the globe, Mapquest provides you with a route to Timbuktu, and Google finds just the obscure web page you have been looking for, you can thank the efficient algorithms designed to solve these problems. The search for efficient algorithms, for a variety of practical problems, still comprises much of the work done in theoretical computer science. These tend to be stylized and abstract, and often much more work is needed to actually implement them on real computer systems.

Perhaps a word is in order about *in*efficient algorithms. Imagine the problem of finding a short route from A to B in a map of a 100 cities (called the "Shortest Path Problem"). One sure (and tedious) way would be to enumerate all routes starting with A and ending with B, computing the distance for each. The number of such ways far exceeds the number of atoms in the universe, and the time to perform it (on the fastest) computer far exceeds the time since the big bang. Obviously, this is not how Mapquest does it! They use a much faster algorithm, which of course is fast on *any* given input map, even with a million cities.

But what about a related task: finding a short route from A to B, traversing every city *exactly once* (this famous problem is known as the "Traveling Salesperson Problem"). Here too the brute-force, inefficient algorithm of enumerating all possibilities would work. But is there a better algorithm? Unlike for the Sortest path Problem above, here we have no idea! And you can be sure that no computer you buy will solve The Traveling Salesperson problem (or other, far more desirable ones) for you – *regardless* of the speed of its hardware – the current lack of an efficient algorithm will render a solution inaccessible even for moderate-sized data like a 100-city map!

In short, the class $P$ consists of exactly those problems we either already solve, or could ever hope to solve in practice! Understanding this class, namely understanding the power and limits of efficient computation, is the major task of computer science.

### 3.3 Grains of salt

Let us make two comments of a more sophisticated nature, on some of the imprecisions above, which only a more technical article can remedy. However, rest assured that these do not affect the moral of the story.

First, $P$ is but one measure of efficiency, and there are many more refined ones, which are perhaps more relevant to practice. Notions like *nearly-linear-time* algorithms, and *fast-on-average* algorithms may be needed (and computational complexity does study them!). Also, different computational settings and different measures of efficiency which are beyond the scope of this article may be (and are!) considered. But we stress that even on moderate-sized data, we can only hope to solve problems in $P$. For this to happen, efficient algorithms for these problems must be discovered!

Second, we have sneakily moved here from solving single "tasks" (like proving Fermat's last theorem or designing a cell phone with given criteria), to solving "problems" (like addition or searching for short routes), which are families of tasks of the same nature, with different input data. The latter ("asymptotic") viewpoint is essential for the very definition of algorithms, and to distinguishing between brute-force, inefficient algorithms and clever, efficient ones; only sufficiently large input data will reveal the difference in the performance of a given algorithm. Indeed, problems faced by mathematicians, scientists and engineers are typically "single instance" problems. However, as the previous section should have demonstrated, many can be collected into *classes* of instances (of proof verification, constraint satisfaction, consistency checks etc. in various settings), which together form problems that can be studied asymptotically.

## 4 $P$ versus $NP$ – can creativity be efficiently automated?

Let us write these definitions yet again:

$P$ is the collection of all tasks for which a solution can be efficiently *computed*.

$NP$ is the collection of all tasks for which a solution, *when given*, can be efficiently *verified*.

It is evident that every problem in $P$ is also in $NP$. The correctness of the algorithm generating the solution automatically certifies that solution!

But what about the converse? Is it possible that for *every* problem for which verifying solution is easy, there is a simple program which would generate such valid solutions efficiently? This is the $P = NP$? question! To make formal sense, this question had to wait to the 1970s for the formal definitions of these classes in the above papers. It is remarkable, however, to note that the $P = NP$? question, in quite precise terms, appears already in 1954, in a letter from Gödel to von Neumann (reproduced and translated in the survey [18]) which was discovered only in the 1990s.

Considering the examples above of problems with efficient verification, let us see what a positive answer to this question would mean.

### 4.1 If $P = NP$...

In a very strong sense, this possibility is utopia for the quest for knowledge and technological development by humans.

There would be a short program that, for every mathematical statement and given page limit, would quickly generate a proof of that length, if one exists!

There would be a short program which, given detailed constraints on any engineering task, would quickly generate a design which meets the given criteria, if one exists. The design of new drugs, cheap energy, better strains of food, safer transportation, and robots that would release us from all unpleasant chores, would become a triviality.

And there would be a short program which, given data on some phenomena and modeling restrictions, would quickly generate a theory to explain that data within the modeling constraints, if one exists. Many things that scientists hope to explain, like about how the brain works, the nature of dark matter and dark energy, the structure and function of proteins, etc. could potentially be done in a jiffy!

This all looks fantastic, and undoubtedly will positively revolutionize the world we live in! Even if *abstract* efficient recognition (that we feel is an inherent part of the discovery process above) could me made *concrete* in only a tiny fraction of the problems above, $P = NP$ will be nothing short than revolution. The possibility that $P = NP$ justifies the investment of immense resources into proving it. If proved, far more effort will go to mechanizing the recognition process, further enhancing the impact of such

a discovery.

### 4.2 Why don't we believe it?

The evidence against the mathematical form of $P = NP$ will become clear when we discuss *universality* in the next section. The rest of this section is informal, and certainly specualtive.

Intuitively, most people revolt against the idea that such amazing discoveries like Wiles' proof of Fermat, Einstein's relativity, Darwin's evolution, Edison's inventions, as well as all the ones we are awaiting, could be produced in succession quickly by a mindless robot. We would similarly revolt against the possibility of a monkey spewing out a perfect Shakespeare tragedy.

And indeed, this analogy is in place. People have a strong sense that *creativity* was absolutely essential in these and other discoveries, and will be essential for important future ones. While elusive to define, people feel that *creativity, ingenuity* or *leap-of-thought* which lead to discoveries are the domain of very singular, talented and well-trained individuals, and that the process leading to discovery is anything but the churning of a prespecified procedure or recipe. These few stand in sharp contrast to the multitudes who can appreciate the discoveries *after* they are made.

So how can one explain the creativity which is demonstrated in the examples above, and elsewhere? Some thinkers develop theories whose foundation is a non computational model for cognitive functions. But all evidence shows that the brain, like every other natural process, is an efficient computational device[4]. While science is extremely far from understanding the brain's hardware and software, the laws of physics govern its behavior. And so if any problem was solved by any brain, it was (by definition) an efficiently solvable problem.

Clearly, some minds are stronger than others for performing different tasks. When others solve problems or make discoveries we feel we couldn't have, we call them creative. But our concern here is with complete, universal creativity, rather than sporadic instances of it. We ask about the solvability of *all* tasks affording efficient recognition. If $P = NP$, any human (or computer) would have the sort of reasoning power traditionally ascribed to deities, and this seems hard to accept.

If this were a court case, it is safe to assume that the jury would vote $P \neq NP$ beyond any reasonable doubt. But $P$ versus $NP$ (despite its legal

[4]One can allow quantum machines too, replacing $P$ by its quantum analog $BQP$, without affecting the principles of this paper, but this would complicate other matters and so we avoid it here

sound) is not a court case but a concrete, open mathematical question. We will test the strength of this jury verdict, and the intuition supporting it, in the next section.

### 4.3 And if $P \neq NP$?

Indeed, the jury feels we live in this real world, and not in the unlikely utopia described above. Well, the good news is that all of us scientists, engineers and mathematicians don't need to look for new jobs! Knowledge acquisition and technology will progress as it always had – slowly and painstakingly, with critical dependence on new creative ideas.

But our job security is not the only reason to rule out $P = NP$. The security and privacy of our identity, personal information, finances and interactions, as we have come to experience in the computer and Internet age, are in mortal danger if $P = NP$! Can $P \neq NP$ put us at ease? Possibly, but we need to know more. We will explain these connections, and further refine the world in which $P \neq NP$ into two possible ones, in section 6

## 5 Universality, or, how do we resolve $P$ versus $NP$?

Let us consider another question, "Trunk Packing".

**Question 4**

Given the dimensions of a set of suitcases, and of the trunk of our car, find a way to pack them all in the trunk, if possible.

Why did we descend to considering such a mundane problem, from the sublime heights of profound scientific questions? Bear with me.

For one, it is a problem in $NP$: if anyone shows us how to arrange the given suitcases in the trunk, we immediately recognize that success!

For another, our real life experience tells us that it seems like a difficult problem. Unlike sorting or addition, for which we have efficient methods which always terminate quickly, here no method suggests itself. We often find, even with 5 suitcases, that we try one configuration after another, always to discover a problem with the last suitcase. But observe - with 100 suitcases (or more, as in packing a movers' truck, or arranging electronic components on a microchip - worthy problems which commercial companies face!), the number of possible arrangements is astronomical. Just like searching for a short route through a 100 cities, or the monkey trying to regenerate "War and Peace", enumerating these possibilities is a brute-force

algorithm which will not terminate (even with all world computer cooperating) before the sun burns itself off (that's about 5000 million years). And 100 pieces of data is a *tiny* input (contrast it with the data produced by the LHC experiment).

So is there an efficient algorithm for this problem? Is "Trunk Packing" in $P$?

We don't know the answer, but we will now explain why you should care (and why we actually never left the sublime heights – this problem is anything but mundane!).

**Trunk Packing is in $P$, *if and only if* $P = NP$.**

Well, now that we see it, we realize of course that at least one direction in this equivalence its trivial. We already noted that "Trunk Packing" was in $NP$, so if it not in $P$, clearly $P \neq NP$. But what about the reverse direction? Note what the reverse says: if you find an efficient algorithm to pack your trunk, you will have automatically done so for *all* problems in $NP$. How on earth is it possible that this simple, earthly problem alone captures the difficulty of all the numerous scientific and technological problems above?

The answer is *Universality*. The "Trunk Packing" problem is universal, or in the CS jargon, "$NP$-complete". This means that not only is it in the class $NP$, but that moreover it is as hard as any other problem in $NP$. Formally, any algorithm for it produces an algorithm of essentially the same efficiency, for any other problem in $NP$. Put differently (this may sound sad), the creativity needed to solve packing problems suffices for *all* creative tasks we talked about.

How can this magic work? The answer is *translation* (or in CS jargon, *reduction*) between problems. It is a 3-step process whose structure we now describe. Take any problem in $NP$ (and its efficient verification). For example, take $Q3$. In the first step, the description of Fermat's last "theorem", (together with its logical verification procedure), is efficiently translated to one *instance* of "Trunk Packing", namely to a set of dimensions for each suitcase and the trunk. In the second step, our hypothetical efficient algorithm solves thsi instance and provides the required packing (if it exists). In thet third step, a reverse translation is applied to the arrangement of suitcases just produced, and efficiently converts it to a proof of Fermat's.

The same can be done to any other mathematical claim. It is translated to some instance of Trunk Packing. If no packing exists, the claim is *false*. If it exists, any solution to that trunk-packing isnatce would translate back to a correct proof of that claim. The same can be done to every question, like the numerous ones above in science, engineering, economics and even art, as long as the verification mechanism can be formalized! As you might guess,

the verification mechanism is the main ingredient in providing the efficient dictionary for translating any problem back and forth to Trunk Packing.

The definition of universality (= $NP$-completeness), and the proof (via explicit translation algorithms) that many natural problems (like Trunk Packing) are $NP$-complete, was a turning point in the history of Computer Science. It originates from the above mentioned seminal papers of Cook, Levin and Karp [2, 12, 13]. By now, thousands of such problems are known, across all sciences, whose computational hardness is *the same*. Thus "Trunk Packing" is not special, and we could have used instead problems on protein folding, airline scheduling, network routing or even in string theory. Such a universal phenomena is extremely rare in science, and points again to the fundamental nature of the class $NP$. Naturally, in Computer Science alone, it includes hundreds of real life computational problems for which companies of all kinds have sought (in vein) fast solutions for decades by thousands of highly trained and motivated individuals. Inadvertantly, all were trying to prove that $P = NP$. *For computer scientists, this overwhelming fact is the pragmatic reason to suspect that* $P \neq NP$.

In the absence of proof that $P \neq NP$, $NP$-completeness is the mark of difficulty for a problem. A programmer who discovers that the problem he/she was assigned to solve is $NP$-complete, is well-justified to start looking for approximations or heuristics, knowing (as does her/his boss) that it is unlikely an efficient algorithm to the given problem exists. A mathematician who discovers that it is $NP$-complete to compute a property of a mathematical structure he/she is studying, (should) know that it is unlikely that a characterization of that property exists, and is well-justified to study special cases.

The wealth and variety of these universal problems, and the decades of unsuccessful effort invested by thousands of individuals (with strong intellectual and commercial motivation) to solve them, is a strong case for believing that $P \neq NP$.

Whether this is convincing or not, remember that the $P$ versus $NP$ question is now completely well-defined. To resolve this major problem, one either needs to devise an efficient algorithm for "Trunk Packing" (proving $P = NP$), or to prove that no such algorithm exists (proving $P \neq NP$). As mentioned, this requires deep understanding of efficient computation, the major focus of Computational Complexity Theory. This field has made tremendous progress and is vibrant with new exciting work, but at this point seems to be far from resolving this conundrum. Still, the existing products of this study are all over the computer and Internet world. Obvious examples are the efficient algorithms already discovered, which underly most of the

applications we use daily. Far less obvious is the use of hard problems (for which we expect no efficient algorithms) to enable security and privacy! We turn to discuss now the unexpected usefulness of difficulty.

# 6 Cryptography: security and privacy in the computer age

Cryptography, the age-old art of secret communication, became a major scientific discipline of the widest practical importance around 1980, when the seminal works of Diffie and Helman [4], Rivest, Shamir and Adleman [16] and of Goldwasser and Micali [8] suggested to formally resting it on computational basis. The reader is referred to the monograph and textbooks by Goldreich [6, 7] for (much) more information on the subject.

## 6.1 The integer factoring problem

We begin with our final sample problem, Integer Factoring.

**Question 5**

Given an integer $N$ as input, find a nontrivial factorization of it (if one exists). Namely find integers $K, M$, both larger than 1 and smaller than $N$, such that $K \times M = N$ (if no such factorization exists then $N$ is called a *prime* number.

For example, if the input was 1541, then the (unique!) answer is 23×67. As usual we are interested in an efficient algorithm for this problem. Let us make a few observations about it.

First, "Integer Factoring " is in the class $NP$: given a solution, verifying its correctness requires only multiplication, which as we noted already has an efficient algorithm.

Second, it is not known to be in $P$. The brute-force algorithm for it, trying all possible numbers below $N$ (or even below $\sqrt{N}$) to see if they divide the input, is extremely inefficient, and completely useless even for inputs with merely 1000 digits. Huge efforts were invested in finding faster algorithms (and we shall soon see why). Some were discovered (which are very clever in their use of deep mathematics), but even they are far from being efficient, and still fail at factoring 1000-digit integers.

Third, note a simple corollary to the previous section: if "Trunk Packing" has an efficient algorithm, so does "Integer Factoring". We can efficiently translate any integer factorization problem into a suitcase arrangement one!

The power and mystery of efficient computation, and the difficulty in characterizing it, already manifest themselves in this simple connection.

But can this relation be reversed? Does a fast algorithm for "Integer Factoring" imply one for "Trunk Packing"? In other words, is "Integer Factoring" $NP$-complete? Can it get this "stamp of difficulty" that (lacking a proof that $P \neq NP$) shows it to be as hard as all mathematical, scientific and technological questions discussed above? Actually, we have good reasons to believe it is not $NP$-complete, but it may still be extremely hard computationally nontheless. So is it easy or hard?

Actually, **who cares?** Factoring integers certainly looks like a basic question for a number theorist, but who else should worry if it is difficult or not? Well, **all of us do!** Yes, you too! Let us explain why.

Today, the integrity and security of digital information relies on the (assumed!) difficulty of "Integer Factoring". When you log-in, pay your bills electronically, shop and invest on-line, any petty thief with a quick integer factoring algorithm can steal your identity, and clean your account in a blink. Far worse, large financial institutions and governments rely on the same assumption, thus the existence of such an algorithm in the wrong hands can do huge damage to economic activity, as practiced today!

Remarkably few people are aware that so much rests on the (assumed) computational difficulty of such simple-to-state problem, which requires (besides creativity) only pencil and paper to solve. Moreover, this has been the state of affairs for a couple of decades now! Like nuclear power and genetic engineering, the possibility of harnessing the assumed difficulty of Integer Factoring for technological development made its benefits (in this case, the Internet and e-commerce revolutions) as well as its potential dangers, part of our daily life.

There are at least two reactions to this shocking news. One, intellectual: what's the connection? How can the difficulty of factoring enable such magic as secure e-commerce? And the second, practical: what can be done to better safeguard digital information?

## 6.2 Resting security and privacy on (more) solid grounds

We shall only touch briefly on the intellectual question (which deserves a separate article), mainly relating those aspects that are relevant to the practical, pressing one.

Just like "Trunk Packing" is a universal for the class of all problems with efficient verification, it turns out that "Integer Factoring" is universal for all cryptographic problems, namely those which arise in digital security

and privacy protocols. This understanding (which includes the highly non-trivial task of defining this class) evolved in the 1980's, following the $NP$-completeness revolution of the 1970's. Again, a key element was efficient translations (or reductions). Solutions (called *protocols*) to cryptographic problems were based (often ingeniously) on Integer Factoring in such a way as to guarantee the following. *Any* quick way an adversary had to break the security or privacy of such a protocol, can be translated into an efficient algorithm for factoring integers (which we assume does not exist!).

The famous *RSA* protocol [16], established such a connection for the most basic cryptographic problem: *Secret Communication*. The tall order taken in the modern definition of this problem completely forbids *any* common information between the two parties who wish to secretly communicate! This fits perfectly the digital world and the fact that many pairs of agents who never met (or existed), might need to suddenly exchange secret information (e.g. like your first transaction with Amazon) over the (insecure) Internet, with eavesdroppers listening to all conversations. In plain language, we are looking for a method which allows Alice and Bob, who never met, to engage in a secret conversation in the presence of their adversary Carl, that nevertheless will be meaningless to Carl. Sounds impossible? If "Integer Factoring" is hard, it can be done!

Here is another, perhaps more "impossible" task which is achievable under the same assumption on factoring. It is called *zero-knowledge* proofs. Suppose that you, Alice, discover something, for which you can write down a formal, convincing argument (a proof a famous mathematical conjecture, like the Riemann Hypothesis, is a good example, but a solution to master-level Sudoku puzzle is just as good). You want to convince your friend, Bob, of your achievement. Sure, you can show Bob your solution, which will certainly convince him, but you are anxious (being somewhat paranoic) that Bob may later claim to others that it was *his* original solution, and no yours (this is a basic problem of copyrights). The magical *zero-knowledge* proofs allow you to convince bob of *any* such achievement of yours, in a way that reveals *absolutely nothing* about the solution! Again – you give *no* information, yet Bob is convinced beyond any reasonable doubt that you had a proof (assuming of course you had one – like with normal proofs, you can never fool Bob to believe youhave a proof if you don't). Just imagine the power of this primitive to controlling malicious behavior in cryptographic situations – agents can be forced to prove that they behave as they should. In normal proofs they would be reluctant, since actions often depend on private information, but with zero-knowledge proofs they can do it without violation of privacy!

Here are some other tasks which can be performed under the same assumption, in a completely digital environment (no physical implements whatsoever): contract signing, public bids, electronic ballots, payments, random selection, and more. We most certainly *do not* assume that any party is trusted by all others. Despite these draconian restrictions, protocols relying on the difficulty of factoring ensure the privacy of participants' secrets (not even a fraction of a bit would leak), as well as resilience against cheaters and saboteurs. As mentioned, the set of techniques developed for these different ingenious protocols culminated to give general methods to solve essentially any cryptographic problem, with arbitrary privacy and security constraints (see [9, 21, 10]).

**Why factoring?**

A remarkable feature of the "Integer Factoring" problem holds the key to this magic. *It is the inverse of an efficiently computable function*, "Integer Multiplication". Indeed, let $Q$ and $R$ be prime[5] numbers (say 1000 digits long) and let $N = Q \times R$ be their product. Contrast the two pieces of data: the pair $(Q, R)$, and their product, $N$. Both pieces of data are *equivalent* from a purely informational standpoint: each one uniquely defines the other! But computationally, holding the first is far more powerful than the second: from the pair $(Q, R)$ we can *efficiently* compute the product $N$ (via long multiplication), in a fraction of a second. But holding $N$, and assuming factoring is hard, we'll never figure out the pair $(Q, R)$. This asymmetry is at the heart of using "Integer Factoring" as a basis of cryptography. Indeed, if I randomly pick $Q, R$ and publish their product $N$ for the whole world to see, I can use this knowledge of the factors to solve problems which depend on $N$, that no one else without this knowledge can solve. In particular, I can decipher sercret messages whose encoding depends on $N$, thereby creating a secret communication channel using which anyone can send me private messages which noone else can read. Now if everyone does what I just did, a magical network of secure communication is erected without any prior joint knowledge between communication parties or any hardware!

The careful reader may have noticed a crucial technical point about the nature of hardness of Integer Factoring required by such application, which we have not addressed before. If we want these *randomly generated* instances of factoring to be hard, we need Integer Factoring to be *typically* hard, and not just on rare instances (it may well be the case that Trunk Packing is typically easy, and only rarely requires trying almost all possibilities).

[5]Again, primes have no nontrivial divisors. We choose primes here so that the factorization of $N$ will be unique.

This notion of *average-case* hardness is also well developed in computational complexity, but we shall not elaborate further on it here.

Are there other problems with such remarkable properties as Integer Factoring, which can serve as universal foundations for cryptography? We'll discuss this in the next subsection.

### 6.3 Inside $P \neq NP$

What if "Integer Factoring" turns out to be easy? How likely is the possibility that an efficient method for it will be discovered tomorrow? And rest assured (or alternatively start panicking) – huge efforts are made to do it, as the stakes are so high!

Well, the answers again are in the domain of computational complexity. Here are some possibilities, indeed, viable research directions pursued by the Computational Complexity community. In the interest of keeping e-commerce alive, it would be nice if either of these succeeds.

**Best:** prove that "Integer Factoring" is not in $P$. This would flatly (and mathematically) rule out an efficient algorithm for factoring, certifying its assumed hardness. Note that such a result would imply $P \neq NP$, which seems extremely difficult, so no one expects this to happen soon (even if indeed factoring is hard).

**Second best:** prove that "Integer Factoring" is $NP$-complete, namely equivalent in complexity to "Trunk Packing" and thus to all those thousands of problems, for which our confidence in their hardness is far stronger. As it happens, techniques have been developed to show that such a result is unlikely (even if factoring is hard), but not impossible as far as we know.

**Next:** Find other "cryptographically universal" problems. This class is actually referred to as *one-way* functions[6], to capture the asymmetry of computational power needed to compute the function and its inverse. Any such universal problem would do; if e.g. "Integer Factoring" ends up being easy, we can alternatively rest the security and privacy of digital information on the (assumed) hardness of any other one-way function (we note that proving this was an enormous intellectual achievement!). Thus finding other one-way functions is an extremely important research direction, of (as we shall soon see) deep scientific interest to boot. At this point, this "universality class" seems much much smaller (or at least more elusive) than $NP$-complete problems. So far, only a handful of different alternatives to

[6]Actually, Integer Factoring is a one-way function with a crucial extra property called "trap-door", but for this article we would not make this further important distinction

"Integer Factoring" have been suggested, and almost none of them are used practically.

In the discussion above we assumed that the hypothetical world in which $P \neq NP$ contains one-way functions, and e-commerce thrives with them. However, it is also possible that there are no one-way functions, and no e-commerce as we know it either!

**And if no one-way functions exist?**

We have elaborated on the huge positive implications of the existence of one-way functions. Indeed, they compensate us at the loss of the $P = NP$ utopia and its far grander promise. What about the possibility that they do not exist? Does it hold any positive prospect (to compensate for the loss of e-commerce)? How likely is it? The attempts to answer these again requires a high-level, scientific formulation of one-way functions.

When we ask for a function which is easy to compute, but hard to invert, we are asking for an efficient process that is not efficiently reversible. In this formulation, we know exactly where to look for one-way functions: all around us!

*Every natural process can be viewed as computational!.*

Natural processes constantly evolving input data (current state of the universe) following the laws of nature to produce the output (next state of the world). Thus weather computes itself, DNA computes proteins, the cell processes these, the brain processes its sensory input into actions, seashells and coral grow into their remarkable formations, schools of fish and flocks of geese migrate, stars burn and galaxies rotate, snowflakes form, snow and rain and apples fall on scientists heads and on and on and on.

A fundamental object of science is to understand the laws governing these computations, and a fundamental test of this understanding is the ability to predict them. Namely, being able to compute, given the input data (current state), the outcome of the process (next state). And the processes we study are, by and large, efficient. When we have understood any such process, we have a candidate one-way function. The current level of scientific understanding provides numerous such candidates! Can all of these efficient processes be reversed?

If there are no one-way functions, the answer is positive. Given the current state, we can always efficiently produce a possible *previous* one. Far more (and proving this requires much more work), the same assumption implies that a *typical* previous state can be efficiently computed from the current one. In short, we can go back in time and find explanations for phenomena we are observing. Indeed, the ability to find typical "explanations" will enable finding ones of minimum entropy (in a well-defined sense), thus

(by Occam's razor), the best explanations. This can have a revolutionary impact in finding meaning in huge sets of data, like astronomical data, biological data, or the output of the LHC experiment. Note that we already saw this as a consequence of $P = NP$, and now we see that even if $P \neq NP$, but one-way functions do not exist, this powerful tool of scientific discovery is still available.

# 7 Conclusions – more challenges of computational complexity

Let us end by noting that we have demonstrated here the deep philosophical, scientific and technological aspects of only two central questions of Computational Complexity: "does $P = NP$?" (can creativity be efficiently automated?) and "do one-way functions exist?" (can every efficient process be reversed?). Let us mention a few others, all of which have similar depth and significance, whose study has already gained us remarkable, and often surprising understanding, but resolution still eludes us.

- What is the power of randomness in computation? Can coin tosses speed it up (as they seem to, in numerous probabilistic algorithms)? And if so, where (in the real world) can we get such perfect randomness? If imperfect randomness exists in nature, can we "purify" it efficiently?
- What is the power of quantum computation? Can classical computers efficiently simulate quantum phenomena? Or can quantum phenomena like superposition speed up classical computation? If so, can sufficiently accurate and decoherence-resistant quantum computers be built?
- What is the power of parallel computation? Can 100 computers solve every problem 100 times faster than one? 10 times faster?
- Are time and space interchangeable? Can we always save on one by spending more of the other?
- What is the power of distributed computation? Which global properties can be obtained by simple, local interactions on a large network?
- What can machines learn? Which algorithms best prepare robots to cope and even excel in an unknown and unexpected world? Can we do better than a baby?

- What happens when we integrate economics in computation? Which incentives can be given to private individuals to induce joint efficient and fault-free computation?

In all these questions, the focus on algorithmic efficiency has suggested new, valuable definitions, or revealed remarkable new structure and properties, of many notions that have been studied for ages, such as randomness, learning, knowledge and proof. And we've only just begun! On top of all its obvious past and future impact on technology, the quest to understand and classify what is efficiently computable thus far transcends the realm of computer science, and stands as a central quest of the scientific agenda.

## Acknowledgements

I wish to thank Scott Aaronson, Sanjeev Arora, Freeman Dyson, Lance Fortnow, Oded Goldreich and Gil Kalai for valuable comments on earlier versions of this paper.

## References

[1] Alan Cobham, The intrinsic computational difficulty of functions. In *Proc. Logic, Methodology, and Philosophy of Science II*, North Holland, (1965), 24–30.

[2] Steve A. Cook, The Complexity of Theorem-Proving Procedures, *Annual ACM Symposium on Theory of Computing* (1971), 151-158.

[3] Steve A. Cook, "The $P$ vs. $NP$ Problem", *CLAY Mathematics Foundation Millenium Problems*, http://www.claymath.org/millennium

[4] Whitfield Diffie and Martin Hellman, "New Directions in Cryptography", *IEEE Transactions on Information Theory*, vol. IT-22, (1976), pp: 644-654.

[5] Jack Edmonds, "Paths, Trees, and Flowers", *Canadian Journal of Mathematics* **17** (1965), 449-467.

[6] Oded Goldreich: Modern Cryptography, Probabilistic Proofs and Pseudorandomness, Algorithms and Combinatorics series (Vol. 17), Springer, (1999).

[7] Oded Goldreich: Foundation of Cryptography (in two volumes: Basic Tools and Basic Applications), Cambridge University Press, (2001) and (2004).

[8] Shafi Goldwasser and Silvio Micali, "Probabilistic Encryption". Journal of Computer Systems and Science, 28, 2, (1984), 270-299.

[9] Oded Goldreich, Silvio Micali and Avi Wigderson, "Proofs that Yield Nothing but their Validity, or All Languages in NP have Zero-Knowledge Proof Systems", *Journal of the ACM*, Vol. 38, No. 1, (1991), 691-729.

[10] Oded Goldreich, Silvio Micali and Avi Wigderson, "How to Play Any Mental Game", *Proc. of 19th STOC*, ( 1987), 218-229.

[11] Russell Impagliazzo, "A personal view of average-case complexity", *Proc. of the 10th IEEE Annual Conference on Structure in Complexity Theory* (1995), 134-147.

[12] Leonid A. Levin, "Universal search problems", *Problemy Peredaci Informacii* **9** (1973), 115-116. English translation in *Problems in Information Transmission* **9** (1973), 265-266.

[13] Richard Karp, "Reducibility among combinatorial problems", *Complexity of Computer Computations* R. E. Miller and J. W. Thatcher (eds.), Plenum Press, (1972), 85-103.

[14] Christos H. Papadimitriou, *Computational Complexity*. Addison Wesley, (1994).

[15] Michael O. Rabin, "Mathematical theory of automata". In *Proceedings of the Nineteenth ACM Symposium in Applied Mathematics*, (1966), 153-175.

[16] Ron Rivest, Adi Shamir, Len Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". *Communications of the ACM*, Vol. 21 (2), ( 1978), 120–126.

[17] Michael Sipser, *Introduction to the Theory of Computation*. PWS (1997).

[18] Michael Sipser, "The History and Status of the $P$ versus $NP$ Question", *STOC* (1992), 603-618.

[19] Avi Wigderson, "$P$, $NP$ and Mathematics - a computational complexity perspective", *Proceedings of the 2006*

*ICM (International Congress of Mathematicians)*, (2006), http://www.math.ias.edu/ avi/PUBLICATIONS/MYPAPERS/W06/w06.pdf

[20] Freek Wiedijk (Ed.): *The Seventeen Provers of the World*, Lecture Notes in Computer Science 3600, Springer (2006).

[21] Andrew C. Yao. "How to generate and exchange secrets", *Proceedings of the Twenty-seventh IEEE Symposium on Foundations of Computer Science*, (1986), 162–167.