The Theory of Computing: A Scientific Perspective

Oded Goldreich[*] Avi Wigderson[†]

June 8, 2001

Abstract

We provide an assessment of the Theory of Computing (TOC), as a fundamental scientific discipline, highlighting the following points:

- TOC is the science of computation. It seeks to understand computational phenomena, be it natural, man-made or imaginative.
- Research in TOC has been extremely successful and productive in the few decades of its existence, with continuously growing momentum. This research has revolutionized the understanding of computation and has deep scientific and philosophical consequences, which will be *further* recognized in the future. Moreover, this research and its dissemination through education and interaction has been responsible for enormous technological progress.

[*]Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, ISRAEL. E-mail: `oded@wisdom.weizmann.ac.il`

[†]Institute for Advanced Study, Princeton, NJ. E-mail: `avi@ias.edu`

Contents

1 Introduction

The revolutionary impact of Computing Technology on our society does not necessarily facilitate the appreciation of the intellectual contents of the Theory of Computing (TOC). Typically, people are so overwhelmed by the wonders of the computing technology that they fail to wonder about the theory underlying it. Furthermore, they tend not to think of computing in general terms but rather in the concrete terms in which they have lastly encountered it. Consequently, the intellectual contents of the Theory of Computing is rarely communicated and rarely understood (by non-specialists).

Our aim is to help to redeem this sour state of affairs and try to communicate the intellectual contents of the Theory of Computing. But before doing so, we explicitly state the philosophical beliefs that underly our view of science in general.

1.1 Culture, Science and Technology

The search for truth and beauty is the essence of civilization. Since the Renaissance, the search for truth takes the form of (or is called) Science. Technology is an important by-product of the scientific progress, not its *raison d'etre*. Furthermore, philosophical reasoning as well as experience show that technology is best served by a free scientific process; that is, a scientific process which evolves according to its own intrinsic logic and is not harnessed to the immediate technological needs. Such free scientific process evolves by formulating and addressing intermediate goals which are aimed at narrowing the gap between the ultimate goals of the discipline and the understanding achieved so far.

It is ironic that as the contribution of science to technology becomes wide-spread, a popular demand arises to have more. Namely, the success of science and in particular the benefits of its technological by-products causes the populace to turn against science (in the form of demands that science deliver even more consumable commodities). Still, one has to oppose these demands. Science is to maintain its *autonomy* which is correlated to its success. In the long run, this is also the best way to serve technology.

Technology evolves mostly via applied scientists and engineers who use the scientific knowledge they have acquired and their own creative forces to the development of specific applications. Contrary to popular beliefs, the most important contributions of science to technology do not stem from the harnessing of scientists to engineering tasks, but rather from the fact that scientists instruct and enrich the thinking of these engineers. The education of engineers does not reduce to the acquisition of information. Its more important features are the development of conceptualization and problem-solving abilities. The conceptual frameworks of the discipline are offered to the student and the better these frameworks are the better an engineer he/she may become. This form of education is most effective when done by good scientists who enjoy the freedom to pursue their own research interests.

It is important to note that the nature of the process by which science effects technology makes it very hard for the laymen, and sometimes even the expert, to trace a technological breakthrough to its scientific origins. Almost always these breakthroughs depend on the conceptual scientific framework and very often they utilize specific discoveries which were considered totally impractical at the time of discovery (e.g., complex numbers and electricity).

1.2 Individual scientific disciplines

The scientific disciplines are defined by the questions they address. For example, the formative questions of Biology refer to (structural and operational) properties of *living beings*, those of Chemistry refer to (structural and operational) properties of (natural and artificial) *substances*, and those of Physics refer to (motion and interaction) properties of *matter and energy.* (Jumping ahead, we identify the formative questions of the Theory of Computation as referring to general properties of *computation* be it natural, man-made, or imaginary.)

The *importance of a discipline* is determined by the nature of its formative questions. The more fundamental these questions are the more important the discipline is. Educated laymen and certainly scientists can usually assess how fundamental major scientific questions are.

The *success of a discipline* is measured by the progress it achieves on its own formative questions. To measure the amount of progress one has to understand the questions and the state of knowledge of the discipline with respect to these questions. This usually requires the understanding of experts, but can be conveyed to scientists of other disciplines.

Neither the importance nor the success of a scientific discipline can be measured by the impact of its current discoveries on technology (or on other disciplines). If the discipline is indeed important and successful such impacts are likely to follow. However, rarely will this impact be linearly related to the scientific progress in the discipline.

Individual scientific disciplines do not exist in a vacuum. The healthy evolution of a scientific discipline is sensitive to *scientifically relevant* inputs from other disciplines as well as technological developments. We wish to stress that the influence of these inputs is determined by the disciplines internal logic and inherent goals and that such influences are vastly different from non-inherent suggestions (e.g., that in order to increase funding and/or employment opportunities the discipline should pursue alternative directions).

1.3 Theory of Computing: A Scientific Discipline

The Theory of Computing aims at understanding general properties of computing, be it natural, man-made, or imaginary. Most importantly, it aims to understand the nature of efficient computation. Following are teasers for four concrete topics which will be discussed in greater length in Section 5.

One key question is *which functions can be efficiently computed?* For example, it is (relatively) easy to multiply integers, but it seems hard to take the product and factor it into its prime components. In general, it seems that there are one-way computations, or put differently *one-way functions*: Such functions are easy to evaluate but hard to invert. Do one-way functions exist? We don't know, though we believe they do exist, and can relate this belief to other important questions.

A related question is that of the comparable difficulty of *solving problems versus verifying the validity of solutions.* We believe that some problems are much harder to solve than to verify the validity of a solution for them. However, we don't know this to be a fact either. Still, we know of many problems which are hard to solve, provided that the above belief is indeed valid. For each of these problems, an efficient solving method would imply an efficient solving method for each problem for which verifying validity of solution is easy.

The Theory of Computing provides a new viewpoint on old phenomena and concepts. For example, a computational approach to randomness leads to the conclusion that randomness can be expanded almost arbitrarily. Likewise, a computational approach to proofs leads to the conclusion that obtaining a proof to a statement may not teach you anything beyond the validity of the statement.

The Theory of Computing is also concerned with finding the most efficient methods for solving specific problems. To demonstrate this line of research we present a method for multiplying numbers which is more efficient than the simple method learned in elementary school.

2 On the fundamental nature of TOC and its success so far

The Nature of Efficient Computation and its natural as well as surprising derivatives, is the formative question of the Theory of Computing (TOC). We consider this question to be one of the most fundamental scientific questions ever asked. Unfortunately, the fundamental status of this question is usually disregarded due to its immediate technological impact.

We feel that both the fundamental nature of the questions of the Theory of Computing and the success of our community in engaging these questions (up to this very day) are evident. To be on the safe side, here is some evidence.

An excellent demonstration of the the fundamental nature of TOC is provided by the impact of NP-completeness on other sciences. Papadimitriou lists about 20 diverse scientific disciplines that were unsuccessfully struggling with some of their internal questions and came to recognize their intrinsic complexity when realizing that these questions are, in some form, NP-complete. According to his bibliographic search, NP-completeness is mentioned as a keyword in about 6,000 scientific articles per year. How many scientific notions have had such impact?

More generally, TOC has established a direct relationship between structural and computational complexity. Efficient algorithms are discovered almost only if tangible mathematical structure exists. This connection has already benefited mathematical progress in many areas such as Number Theory, Algebra, Group Theory and Combinatorics, where on one hand a need for efficient algorithms existed, and on the other hand the search for them has generated structural results of independent interest.

Actually, we tend to forget the revolution in problem-solving introduced by the TOC treatment of algorithms. This revolution consists of the explicit introduction of the concept of an algorithm and the measures for its efficiency, the emphasis on data representation and organization, the general techniques for creating algorithms for classes of problems, and the notion of reductions between problems. Needless to mention the impact of all these on computer practice, but we wish to stress the impact on any kind of problem solving.

The TOC has drastically changed the perception of knowledge and information. Specifically, the TOC stresses that different representations of the same information may not be *effectively* equivalent; that is, it may be infeasible to move from one representation to the other (although a transformation does exist). In this new world, publicly available information may be unintelegible. All of Modern Cryptography is based on this Archimedes' point, and its scientific and technological impact are well known. Here we wish to suggest that this revolution applies not only to computer systems but to any aspect of human interaction in which privacy and fault-tolerance are important concerns.

The TOC has introduced totally novel ways of understanding and using randomness. The probabilistic algorithms developed within the TOC use randomness in many varied sophisticated ways. The applicability of randomized procedures for solving tasks from different domains such as number theory, optimization and distributed computing is amazing. Moreover, the growing study of derandomization has lead to derivation of better deterministic algorithms from probabilistic ones.

Combining randomness and interaction lead TOC to create and successfully investigate fascinating concepts such as interactive proofs, zero-knowledge proofs and Probabilistically Checkable Proofs (PCP). Each of these concepts introduces a deep and fruitful revolution in the understanding of the notion of a proof, one of the most fundamental notions of civilization. Furthermore, these revolutions bore fruits which reached far beyond the realm of proof systems. For example, work on PCP lead to the first breakthrough in the understanding of the hardness of approximation. This is but one incredible demonstration of the how probabilistic thinking leads (very indirectly and non-trivially) to fundamental understanding of totally non-random phenomena.

In addition, combining randomness and complexity, TOC has generated meaningful notions of pseudorandomness. Computational hardness yields pseudorandom generators: using "one-way" functions, randomness can be "stretched" in an almost unlimited way as far as efficient observations are concerned. This yields the stunning (to most scientists) conclusion that if their Monte-Carlo algorithm (estimating perhaps a numerical integral or simulating a physical process) behaved differently on sequences produced by such generator, than on genuine random sequences, then they have discovered an efficient factoring algorithm! Totally different pseudorandom generators which TOC discovered can fool any space limited algorithm. Since all standard statistical tests have such implementations, this is great news to Statisticians, Physicists, and most Social Scientists who use such tests on everyday basis. Namely, the results of all their experiments are guaranteed to hold even if they replace all their random choices by pseudorandom choices produced by from tiny random seed.

TOC has gained considerable understanding of organizing work on huge systems of many components. The study of parallel algorithms resulted in amazing ways to get around "inherently sequential" tasks. Subdividing work to smaller chunks in efficient and balanced ways is taking place not only in computer systems but in many organizations, and the insights gained by TOC are avail to them too. A different kind of parallel computing arises in settings where the information is distributed among the components of the system. TOC studies of such distributed environments resulting in models and methods of consistency, recovery, knowledge, synchrony and decision making, are relevant not only to (distributed) computer systems but also to economics and other social sciences.

The organization and availability of information was always a major part of civilization, and in particular science and technology depend on it. The models and solutions developed by TOC for such problems not only resulted in computer systems that would do it for people, but in the very way people and institutions have to think about information. The amazing new abilities to handle huge masses of data increase, rather than decrease, the human decisions on what they want to be stored, what access patterns they want to allow and disallow, what should be retrieved quickly and what can take longer, etc. The theoretical understanding enables to formalize their demands, and enable programmers (who should understand the algorithms and data structures as well) either to satisfy these demands or to explain why they are impossible to achieve.

Likewise, some of TOC's insights to performance analysis, the minimizing and balancing of several resources, are of universal applicability. One example is the notion (and techniques) of competitive analysis, whose applications range from operating systems to information compression (Lempel-Ziv) to emergency services to stock-market investments. More generally, asymptotic analysis has taught us that structure is often revealed at the limit. The adversarial point of view developed for worst case analysis (both of inputs to algorithms and behavior of distributed systems) has taught us a similar lesson: structure is often revealed under the worse circumstances and may be obscured by unjustified assumptions on "typical behavior". Such structure often leads to better (in every respect) theoretical and practical solutions.

Finally, let us mention that that many inter-disciplinary scientific activities involve and further seek the participation of TOC members. These include the different "neurocomputational" groups (encompassing brain models, learning, and neural networks, involving physicists, biologists, psychologists) and "rational behavior" groups (encompassing economy, ecology, evolution, competition, and decision making, involving economists, statisticians, psychologists and mathematicians). They want TOC to be there since they have recognized the universal value of the problems TOC deals with and the understanding TOC has obtained so far, and in particular their relevance to these areas.

Clearly, lack of space, time and knowledge prevents us from going on. Still, the massive list above illustrates the fundamental nature of our endevours from the scientific point of view. But they are fundamental also from two other important viewpoints. One is the philosophical viewpoint, which has dealt with many of the notions and questions above for centuries, and which receives a fresh, radically different perspective (namely the computational one) from TOC. As an example consider the question of P vs. NP vs. CoNP. Some tend to think of it is a mere technical question and miss its deep philosophical significance: Understanding the relation between the difficulty of solving a problem to the difficulty of verifying the correctness of the solution, to the difficulty of proving that no solution exists. Additional examples are the TOC perceptions of the notion of a proof, its view of randomness, and its emphasis on the importance of specific representations. The second viewpoint is the potential contribution of TOC to the general education and enrichment of humanity. Many notions, problems and even some of the solutions TOC has produced are available for understanding (in nontrivial levels) by laymen. We have successfully tried to explain some of them to elementary school kids (and indeed we foresee some of them taught and used as teaching paradigms in grade and high school). Few sciences (which existed for many centuries) can compete on these grounds with what TOC achieved in a few decades.

To summarize, this section illustrated the fundamental importance of TOC as well as its success. As for the latter point, let us stress that the achievements sketched above are more or less equally spread over the last 30 years, and many are very recent. Indeed, the rate of progress done by TOC in these years is astonishing and there is no inherent reason for this progress to stop.

3 On the impact of TOC on Technology

While we rejected technological impact as a measure of importance and progress of a scientific discipline, the enormous impact of TOC research on technology should not be made a secret. We are far from experts regarding this impact, still there are a few points that even we can tell. We hope and believe that a much better treatment will be given in the future by more qualified colleagues.

The most important impact of TOC on Computer Science and Technology stems from the fundamental goals of TOC. In its endevour to understand the nature of computation, TOC created general abilities to conceptualize, model, unify, solve and analyze computational mediums and problems. The effects of this understanding are present in essentially every working system and algorithm on earth. Without them the computer revolution, which has changed life on this planet in a fundamental way and will continue to effect it at increasing speed, would simply not be possible! Indeed, they are the very reason that theory courses are mandatory for all undergraduates in computer science departments. They are the reason that most applied computer science courses are not a mere collection of ad-hoc tricks and are thus suitable to be taught in universities. They are the reason that the originators of technological breakthroughs, as well as all engineers and programmers, can actually think, talk, present and evaluate their ideas. Some critics may say that these understandings were achieved long ago, and there is no need for further "refinements". This is contradicted by many technological advances which have resulted (and will continue to result) from *recent* developments of such understandings regarding, for example, parallel, distributed, interactive, secure and fault-tolerant computation. Many such developments were achieved by special interest groups within TOC, who took on to study in depth such models and algorithms. Their specialized conferences, which are a relatively recent phenomena, often enjoy the active participation of more applied scientists, who have both easy access to this knowledge as well as a forum to influence its direction.

It is crucial to recognize and communicate the fact that most of this understanding resulted not from attempts of solve a concrete problem under particular technological constraints. Rather, it came from generalizing the problems and abstracting away unnecessary technological details to the point that enables finding structures and connections to other knowledge. Only then could applied scientists and engineers, who had both the theoretical understanding as well as the mastership of the specifics of the technological task, fuse them together to a successful practical object. The value of this approach has many examples, and we discuss only one.

By far the largest impact computers had on humanity is the Internet. Here are a few key theoretical developements, mostly done much before the internet was even conceieved, that were absolutely essential to its deployment and success (but by no means undercut the enormous contribution of the practical side of CS and the Industry to the Internet revolution).

- Cryptography is the key to people trusting the internet, for their privacy, and their ability to conduct a variety of transactions securely. In brief, Cryptography guarantees the huge *economic potential of e-commerce* on the Internet.
- Distributed Computation is the key for the various protocols and algorithms making sure packets are routed quickly and reliably. In brief, it delivers the *effciency of communication* on the Internet.
- Algorithms and Data Structure drive the fantastic speed of information processing and retrieval. In brief, they deliver *search*, which is the main activity on the Internet.

In general, one should advocate the value of abstractions which address some fundamental aspects of an important problem (even if they seem not to address all aspects), and warn against the shortsightedness captured by dismissing such abstractions as irrelevant. The study of such an abstraction is more likely to yield fundamental insights than the study of the "real problem" (assuming such a creature exists – actually there is never one real problem but rather many different related real problems and what these have in common may well be the dismissed abstraction). Only later will people, with a concrete application and technology in mind, be able to fine-tune the theoretical understanding to their needs. (This in itself may require significant research and implementation, that was and is taking place by computer scientists and engineers, and which resulted in so many successful technological developments.)

It is equally important to recognize and communicate that it was the freedom and time given to TOC researchers to pursue these general directions, in real attempt to understand novel computational media, that resulted in such progress – quite often in surprising and unexpected ways.

One can illustrate the point above by numerous examples. We prefer to give two very recent examples whose technological and practical effects are imminent and yet to come. So far their "practicality" is demonstrated by a major leap in the algorithmic understanding of major problems. This leap is rooted in developments of complexity theory which, at first and for a long time, seemed totally irrelevant to the latter or any other algorithmic task. Such leaps are frequent in our field, and are due to the freedom of pursuing scientific intuition, as well as to the strong communication and information exchange between the various subareas of our field.

- **The Euclidean TSP Algorithm.** A few years ago Sanjeev Arora announced a polynomial time approximation scheme for the Traveling Salesman Problem (and a host of other combinatorial optimization problems) in the plane. The problem itself was a major object of study in our field for decades. The failed attempts to find such approximation scheme resulted in fundamental contributions to NP-completeness, probabilistic analysis, approximation algorithms and mathematical programming. It also resulted in enormous efforts to understand the relative power of various heuristics.

The techniques present in the algorithm of Arora were available decades ago! Why was it only found now? While this is a source of speculations, Arora himself tells how he came about it. The algorithm arose from his attempts to generalize the inapproximability results of metric TSP to Euclidean TSP, attempts which revealed to him the extra structures of the Euclidean case. These attempts were based on the surprising connection of PCP proofs to hardness of approximation. In turn, these "mysterious" proofs arised from abstract results like MIP=NEXP (relating "clearly impractical" complexity classes). Moreover, the MIP model of multi-prover interactive proofs was suggested by Shafi Goldwasser as a generalization of interactive proofs (themselves the outcome of amazing developments). Needless to say that Goldwasser did not think of approximation algorithms when she suggested the new model.

- **Efficient Error Correction.** Again, a few years ago Dan Spielman discovered a *linear-rate* code which has asymptotically optimal (i.e., linear time) encoding and decoding algorithms. This central problem of communication, that originated with Shannon half a century ago, has attracted the best minds in Information Theory, Mathematics, Electrical Engineering and Computer Science, and has resulted in beautiful and important theory. Still, this major problem, resolved by Spielman, was beyond reach.

  The construction of Spielman closely mimics the construction of a superconcentrator. This object was not available to most scientists working on this problem, and Spielman learned about it from Complexity Theory. The superconcentrator was invented in TOC, by Valiant, in his attempts at one of the quinticential impractical problems – proving circuit lower bounds. Failing to do that, Valiant turned to an even more impractical problem – to show that this particular attempts will necessarily fail! Here he was successful. He (noconstructively) exhibited the existence of expanders, and used them as building blocks of linear size superconcentrators. A deep and beautiful mathematical theory developed, motivated by the explicit and efficient construction of expanders, which effected diverse areas of TOC. More to the point of this subsection, indirectly and through much further work, derivatives of the study of expanders became extremely relevant to technological development concerning communication networks and protocols for a variety of parallel and distributed architectures.

The amazing scientific consequences and the surprising practical implications which sprouted (and will continue to grow) from the totally abstract and impractical proposals of Goldwasser and Valiant in the examples above, well illustrate the richness and unity of our field. Such connections seem to be more common in TCS than in other fields.

4 On the impact of TOC on other sciences

In the short time of its existence, TOC has had an unprecedented effect on other sciences. This has taken at least three forms.

- **Algorithms.** Many sciences use heavy computation for their research, mainly for simulation and analysis. The advances in fundamental algorithms in TOC, on data structures and general techniques are essential for them to understand, so as to optimize their computational resources. The impact of these on the rate of progress in these sciences cannot be underestimated. Moreover, sometimes such disciplines generate a particular type of problems for which the general algorithmic knowledge does not suffice. In some cases where these problems raised sufficient scientific interest (perhaps luckily timed with internal developments), TOC was quick to pick up and study its natural computational structure. Two such superb examples are the great advances TOC has made in understanding and analyzing random walks, so often at the base of simulations in Physics, and its contributions to number theoretic and algebraic algorithms. Finally, the success of the Human Genome Project, was partially based on algorithmic progress on problems related to sequencing and other computational biology induced problems of massive information processing. Much more essential will algorithms be for the real challenge of understanding the structure and function of genes and proteins.

- **Natural Computational Models.** Nature computes! While this was observed long before computer science existed, TOC supplied the mechanisms to model, discuss and explain these phenomena. A recent challenge directed by TOC towards Physics is whether a Quantum Computer can be built? But even without the demonstration of the excessive power of the Quantum Computer model (e.g., Shor's polynomial-time Quantum algorithm for factoring), we speculate that complexity may be the right way of thinking about decoherence of a quantum mechanical system. The brain is another computational device whose understanding seems to be extremely far, but to which our unique contributions in neural networks and computational learning are providing important stimulation. Valiant's book "Circuits of the Mind" is the first serious attempt in any of the sciences studying the brain to relate the what we know of the "hardware" in our brain, to the computational complexity of the "functions" it manages to perform. Understanding the complexity of cognitive tasks, and our ability to perform them is a great challenge to TOC.

- **Universality of TOC notions.** As pointed out in Section 2, the unique computational point of view of TOC and its conceptual derivatives, has resulted in surprising impact on intrinsic studies of other disciplines. NP completeness, discovered over 20 years ago, has had a sweeping effect. But our view on other notions such as randomness, pseudorandomness, interaction and approximation is only beginning to take effect.

It should be reiterated that the discoveries above has made a fundamental impact on these sciences, and have lead them to reassess their points of view on some basic intrinsic questions and pursue novel research directions. We wish to stress that, having sound tradition and self esteem, these sciences were not (and could not have been) forced to pursue these novel directions by TOC or anyone else. Their choice was based on their scientific understanding of their intrinsic goals. Similarly, the interest of TOC in these problems arose from the understanding of TOC researchers that these problems are relevance to the goal of understanding computation. The amazing success of this impact and the high and growing regard to TOC in these sciences, again, stems from the intellectual freedom in which these interactions arose. Again, even a small fraction of these effects justified the investment so far in TOC.

5 Four concrete topics of investigation in TOC

The following exposition is aimed at laymen, and we hope that it can be understood by such.

Before embarking, we point out that the choice of representation of objects plays a key role in the theory of computing. If you care to talk of multiplying numbers, you should say in what form are these numbers represented. The natural choice, which the Theory of Computing indeed adopts, is a (natural) number is represented as a sequence of decimal digits. (Actually, the common convention is to represent numbers as sequences of binary digits, but the difference between the two conventions is immaterial.)

5.1 One-way functions (do they exist?)

We consider functions which map natural numbers to natural numbers. To simply the discussion, we consider only functions which are one-to-one (i.e., never map two different numbers to the same number) and preserve the magnitude of numbers (i.e., the number of digits in the representation is preserved when applying the function).

A function is called *one-way* if it is (relatively) easy to evaluate but hard to invert. For example, consider the function which maps pairs of prime numbers to their product. The elementary method for multiplying numbers demonstrates that it is relatively easy to evaluate this function. (By the way, more efficient methods for multiplication are known; see below.) However, we do not know of an *efficient* method for inverting the above function; that is, for going from the product back to the prime factors. In fact, the problem of factoring numbers is believed to be hard.

To get some feeling for the plausibility of the belief that factoring numbers is fundamentally more difficult than multiplying them, think of the task of multiplying two 4-digit numbers (for example, 5381 and 6673). Certainly, you can do this using a pen and paper within a couple of minutes. But how about finding the prime factors of a 8-digit number (for example, 51855637)?

Any one-way function can be inverted by trying all possible inverses, but such an exhaustive search is not efficient: To invert the function on a 100-digit number, an exhaustive search will take 10^{100} operations (which will take more time than the age of the universe even using the fastest possible computer ever to be built). For some functions, there are more efficient ways of inverting the function (for example, consider the function which maps an integer to its successor – that is, n is mapped to $n+1$). The question is whether *every* function which is easy to evaluate is also easy to invert. Our belief is that the answer is negative; that is, that they are functions (called one-way) which are easy to evaluate but hard to invert. In case our belief is wrong this would mean that any process can be reversed within an effort which is proportional to the effort invested in carrying it through. Analogies from many disciplines suggest that this cannot be true in general. That is, some processes may be easy to reverse, but there are processes which are hard to reverse.

Trying to prove that one-way functions do exist is indeed within the agenda of the Theory of Computing and so is exploring the consequences of assuming that one-way functions exist. For example, it turns out that "Cryptography" is possible if and only if one-way functions exist (see more below).

5.2 Solving problems versus verifying solutions

When I say a "problem" I mean a general type of a problem for which they are many instances. For example, consider the problem of finding a number which (strictly) divides a given number. In this case the instances are numbers and each instance may have several solutions (for example, 385 is an instance and 5, 7 and 11 are all solutions (that is non-trivial divisors)). There may be instances which have no solution (for example the number 17 has no non-trivial divisors). It is easy to *verify the validity of solutions* to instances of the problem we are discussing here: Given two numbers N and M it is easy to test if M divides N. However, it seems hard to solve this problem for given instances: Recall that we believe that it is hard to factor numbers into their prime components. Thus, if we could always (easily) find a divisor of a given number (or tell if such does not exist) then we could factor. (This claim is not immediate: you may need to apply the divisor-finding method several times, but not too many times...)

In general a problem consists of a set of instances each having a (possibly empty) set of solutions. With respect to such a problem we consider two computational tasks:

solving: given an instance of the problem find a valid solution or indicate that no such solution exists (if this is indeed the case).

verifying: given an instance of the problem and a candidate solution, determine whether the candidate is indeed a legitimate solution to the given instance.

The big question of the Theory of Computing is what is the relation between the difficulty (or complexity) of the above two tasks. Specifically, whether for each problem for which the verification task is easy also the solving task is easy. This question is known as the "$\mathcal{P}$ vs $\mathcal{NP}$" question: Loosely speaking, $\mathcal{P}$ stands for the class of problems which can be solved easily, $\mathcal{NP}$ stands for the class of problems for which verification is easy, and the question is whether $\mathcal{P}$ contains everything in $\mathcal{NP}$.

Another Example. Suppose you are given a set of Quadratic equation and is asked to find 0-1 values for the variables so that all equations are satisfied. For example, consider the system

$$\begin{aligned} x_1x_2 - x_3 &= 0 \\ x_1x_3 - x_1x_4 + x_3x_4 &= 1 \\ x_1x_4 - x_2x_3 + x_1x_3 &= 0 \end{aligned}$$

You may easily verify that the setting $x_1 = x_2 = x_3 = 1$ and $x_4 = 0$ satisfies all requirements, but it would have taken you more effort to find such a setting by yourself. In general, the verification task is easy (you just substitute variables by their values and do a little arithmetics), whereas the solving task (finding a 0-1 setting satisfying all equations) seems hard. Note that there is an obvious (but *inefficient*!) way of solving the problem: just trying all possible solutions. But this is not feasible if you have a system with many (say 100) variables. The question is whether there exists an *efficient* way of solving the above problem. We believe that no such efficient method exists. Furthermore, we can show that an efficient method of finding solutions to Quadratic equations as above would yield an efficient method for solving any problem in $\mathcal{NP}$ (that is, it would yield that $\mathcal{P} = \mathcal{NP}$). Indeed, the latter statement is interesting and surprising: the fate of the "$\mathcal{P}$ vs $\mathcal{NP}$" question depends on whether it is easy to solve Quadratic equations. Thus, we say that solving Quadratic equations is *NP-complete* (see below).

The belief that $\mathcal{P}$ does not contain all $\mathcal{NP}$. Recall that we do not know whether for each problem for which the verification task is easy also the solving task is easy. That is we do not know whether $\mathcal{NP} = \mathcal{P}$ or not. We do however believe that there are problems for which verification is easy and yet solving is hard (that is $\mathcal{NP} \neq \mathcal{P}$). This belief is based not only on the intuition that solving is generally harder than verifying validity of solutions, but also by a variety of problems (in $\mathcal{NP}$) for which many people failed to find efficient solution-finding procedures.

NP-completeness. There are many problems (the above example is merely one of them) for which we know that an efficient way of finding solutions for the problem would yield such efficient solutions for any problem in $\mathcal{NP}$. Thus, each of the former problems, called *NP-complete*, encompasses the fate of all $\mathcal{NP}$. If an NP-complete problem can be solved efficiently then any problem in $\mathcal{NP}$ can be solved efficiently (that $\mathcal{NP} = \mathcal{P}$). However, our belief that $\mathcal{NP} \neq \mathcal{P}$ implies that no NP-complete problem has an efficient solution-finding procedure. Thus, NP-completeness of a problem is taken as strong evidence that it cannot be solved efficiently.

Indeed NP-completeness is extensively used as an indication of the complexity of problems. Once you are faced with a particular problem which you need to solve and once you have failed to devise efficient solution-finding procedure, you may want to know if your failure is due to your own lack of ideas or to the intrinsic difficulty of the problem at hand. Proving that the problem is NP-complete does provide an indication that your failure is due to something more fundamental than your lack of ideas. That is indeed comforting, but what should you do if you still need a solution? In such a case, having realized that the problem at hand is NP-complete, you should seek relaxations of it, which are good enough for the application at hand, and try to obtain an efficient procedure for solving such a relaxed problem. The relaxation can take the form of restricting the set of possible instances or broadening the set of admittable solutions. For example, if you only need to find 0-1 solutions to a set of linear equations, you should not worry that finding solutions to Quadratic equations is NP-complete: An efficient method for the special case of linear equations does exist! In this case the relaxed problem restricts the set of instances of the original problem. Also, if you are happy with satisfying only half of the given Quadratic equations then there is an efficient method for finding a 0-1 setting which will do the job. In this case the relaxed problem broadens the set of admissible solutions. Thus, NP-completeness told you to look for good enough relaxations of the problem, and can be used as a justification for not solving the original problem. This justification is especially of value if solving the original problem would have been even better.

5.3 Computational View of Phenomena and Concepts

Pseudorandomness. Adopting a computational view of randomness, we call a distribution *pseudorandom* if it is infeasible to distinguish between examples drawn from this distribution and examples drawn from a truly random distribution. Note that two distributions may be very different and yet it may be infeasible to tell them apart. In such a case we consider the difference between them as "non-important" (since nobody can note it within his lifetime, as noting the difference requires an infeasible computation). Thus, our computational view of randomness is behavioristic (it asks how does randomness look to us) rather than being ontological (asking what is the essence of randomness). However, it is the notion of efficient computation which allows such a meaningful and appealing (behavioristic) approach.

More importantly, we may talk of *pseudorandom generators.* These are efficient (deterministic) procedures which once fed with a short random *seed*, output a much longer sequence which is pseudorandom. Thus, pseudorandom generators "stretch randomness": taking a short random seed they produce a much longer sequence which cannot be told apart from a truly long random sequence. To be specific, if you want to produce a 1,000,000 long sequence of random looking digits, it may suffice for you to randomly select 1000 digits and stretch them using an efficient (deterministic) program into a sequence of 1,000,000 digits. Note that the generated sequence is not truly random, yet it looks so to any (computationally-bounded) observer. Since in real-life we are all computationally-bounded, this type of pseudorandomness suffices for all our purposes.

Pseudorandom generators can be constructed provided that one-way functions exist. Actually, this sufficient condition is also a necessary one. Thus, a tight connection is made between computational difficulty (of inverting some functions) and random behavior. Specifically, if computational difficulty does exist in a meaningful way then randomness can be expanded very drastically and so the can be no meaningful measure for the "amount" of randomness. In particular, little randomness may give rise to huge random phenomena and constructs. For example, given 1000 randomly selected digits it is possible to efficiently implement a random function which assigns a random-looking 1000 digit number to every 1000 digit argument. By this we mean that querying this function for its value at, say 1,000,000 places of your choice, you will not be able to distinguish the function from a truly random one.

An application to Cryptography: Pseudorandom generators yield a solution to the problem of securely communicating over an insecure (that is, possibly wire-tapped) channel. Specifically, this is the case since any pseudorandom generator yields a private-key encryption scheme. Such a scheme consists of two procedures, one for encoding and one for decoding. Both procedures utilize a secret key which is assumed to be selected and shared by the communicating parties. Before sending a message, the sender encrypts it using the shared key, obtaining a so-called *ciphertext*. Only the ciphertext is sent over the insecure channel, but a wire-tapper which does not know the key shared by the legitimate parties cannot make any sense of it. Once the ciphertext reaches the legitimate receiver, he/she can read the original message by decrypting the ciphertext using the shared key. Now let us see how to use a pseudorandom generator to establish such an encryption scheme. The key shared by the legitimate parties will serve as a seed to the pseudorandom generator (and thus it is important that the key be selected at random). Messages to be sent are represented as sequences of digits. To send a specific digit secretly, the sender uses the next (unused so far) digit of the pseudorandom sequence (generated by the pseudorandom generator using the key as seed). Say that the message digit is x and the pseudorandom digit is y, the the corresponding digit of the ciphertext will be the least significant digit of $x + y$ (for example, if $x = 4$ and $y = 7$ we send 1 and if $x = 6$ and $y = 2$ we send 8). Decryption is done analogously. Say we have received the digit z and currently use the pseudorandom digit y, then we compute $z - y$ and add 10 to it in case it is negative: for example, if $z = 1$ and $y = 7$ we retrieve $x = (1 - 7) + 10 = 4$ and if $z = 8$ and $y = 2$ we retrieve $x = (8 - 2) = 6$.

Zero-Knowledge. Do proofs teach us anything beyond the validity of the assertion? Our daily life (and especially our school years) tell us that the answer is positive. Typically, convincing us of the validity of some fact the prover (that is, the person convincing us) tell us things we did not know. Adopting a computational view of proofs, we may introduce a meaningful and appealing setting in which proofs exists which yield nothing beyond the validity of the claim the are supposed to vouch for. Such proofs are called *zero-knowledge* since they tell us nothing we did not know (or could not do) if we were to believe the validity of the assertion.

But first we should ask what *is* a proof. The glory given to the creativity required to find proofs, makes us forget that it is the less glorified process of verification which gives proofs their value. What makes gives a proof its value is the existence of an *efficient* verification *procedure* which rejects false proofs while admitting valid proofs. Thus, any (verification) process which has these features gives rise to a "proof system" and, in particular, one may want to consider interactive and randomized verification procedures. Indeed, it turns out that one may be able to verify more facts by employing an interactive and randomized verification procedure (rather than sticking to the traditional perception of proofs as written texts). For example, suppose that a wine expert wishes to convince a non-expert that two bottles of wine are different. Here is what they can do. The (non-expert) verifier will secretly pour wine from the two bottles to (say) 10 different glasses so that each bottle serves 5 glasses. The verifier will randomly permute the glasses, but keep (secret) record of which bottle served which glass. The expert will now be asked to tell which 5 glasses (out of the 10) have wine from the same bottle. If the bottles are indeed different (and if the expert is indeed an expert) then the expert will have no trouble giving the right answer and so the claim will be accepted by the verifier. However, if the two bottles are identical then there is no way of telling the 10 glasses apart and the probability that an expert will guess correctly is quite small (it is one over $\binom{10}{5}$).

The above example illustrates something of the flavor of the computational point of view of proofs. Furthermore, it even has some zero-knowledge flavor: the verifier following the above procedure does not really learn anything new beyond being convinced of the validity of the claim; having poured the wine into the glasses he learns nothing when the expert identifies correctly which bottle served which glass. In general, it has been shown that whatever can be proven via an interactive and randomized process (as above), can also be proven in zero-knowledge.

An application to Cryptography: Zero-knowledge proofs are not merely an intriguing notion, they are a very powerful tool in cryptography. In a typical cryptographic setting parties have secrets and are supposed to take actions based on these secrets. A typical problem is to make sure that the actions taken are indeed correct. This can be demonstrated by revealing the secrets, but zero-knowledge proofs allow to prove this fact without revealing the secrets (and without revealing anything about the secrets).

5.4 The Search for More Efficient Procedures

How would you multiply two numbers? I guess that you would just apply the method taught at elementary school. For example to multiply 45 by 67 you would just write

$$\begin{array}{rcrcl} & & 10\cdot(4\times 7) & + & (5\times 7) \\ 100\cdot(4\times 6) & + & 10\cdot(5\times 6) & & \end{array}$$

Which means that you would do 4 digit-by-digit multiplications, some shifts ("hidden" multiplications by 10, which are indeed easy), and some additions. In general, to multiply two numbers x and y, represented by the digit-sequences $x_n \cdots x_2x_1$ and $y_n \cdots y_2y_1$, respectively, you will turn out using (implicitly) the following equality

$$x \times y \;=\; (\sum_{i=1}^{n} x_i \cdot 10^{i-1}) \times (\sum_{i=1}^{n} y_i \cdot 10^{i-1}) \;=\; \sum_{i=1}^{n}\sum_{j=1}^{n}(x_i \times y_j)\cdot 10^{i+j-2}$$

which means that you would do at least n^2 basic operations (that is, digit-by-digit multiplication). There is however a faster way to multiply (large) numbers. Consider, for example the multiplication of 45 by 67. We have

$$\begin{aligned} 45 \times 67 &= (10\cdot 4 + 5)\times(10\cdot 6 + 7) \\ &= 100\cdot(4\times 6) + 10\cdot(4\times 7 + 5\times 6) + (5\times 7) \\ &= 100\cdot M_1 + 10\cdot(M_3 - M_1 - M_2) + M_2 \end{aligned}$$

where $M_1 = 4\times 6$, $M_2 = 5\times 7$, and $M_3 = (4+5)\times(6+7)$. The last equality does not seem to "make sense" yet you can easily verify that it is correct. But what have we gain by this "strange" equality? One thing is that we only do 3 multiplications (but they may be slightly more complex since we may need to multiply numbers smaller than 19 rather than numbers smaller than 10 (single digits)). This seems little gain, but wait a moment before passing verdict. Suppose you want to multiply two 4-digit numbers. You can represent each number by a sequence of two 2-digit numbers and apply the same trick. That is

$$\begin{aligned} 1234 \times 5678 &= (100 \cdot 12 + 34) \times (100 \cdot 56 + 78) \\ &= 10000 \cdot (12 \times 56) + 100 \cdot (12 \times 78 + 56 \times 34) + (34 \times 78) \\ &= 10000 \cdot M_1 + 100 \cdot (M_3 - M_1 - M_2) + M_2 \end{aligned}$$

where $M_1 = 12 \times 78$, $M_2 = 34 \times 78$, and $M_3 = (12 + 34) \times (56 + 78)$. We may now apply the above to the 3 (two-digit) multiplications we need here, and obtain a procedure involving 9 "basic" multiplications. Generalizing this idea, we obtain a procedure which multiply two n-digit numbers by doing $20n^{\alpha}$ basic operations (that is additions/multiplications of single digits), where $\alpha = \log_2 3 \approx 1.585$. For $n \geq 100$ this is better than the "Elementary-School" procedure (which takes $3n^2$ basic operations). But actually, there are even faster procedures for multiplying two numbers (which do beat the "Elementary-School" procedure for numbers of 20 digits or more).

The above example of a sophisticated computational procedure was taken from the domain of arithmetic, and indeed the study of efficient procedures for arithmetic problems constitutes one area of the Theory of Computation. Yet, there are dozens of other such areas which are occupied with the study of problems arising in other domains.