Invitation to Complexity Theory

Complexity theory provides new viewpoints on various phenomena that were once considered by past thinkers.

By Oded Goldreich

DOI: 10.1145/2090276.2090285

The striving for efficiency is ancient and universal, as time and other resources are always in shortage. Thus, the question of which tasks can be performed efficiently is central to the human experience. A key step toward the systematic study of the aforementioned question is a rigorous definition of the notion of a task and of procedures for solving tasks. These definitions were provided by computability theory, which emerged in the 1930s with the work of Alan Turing (and others). This theory focuses on computational tasks, considers automated procedures (i.e., computing devices and algorithms) that may solve such tasks, and studies the class of solvable tasks.

In focusing attention on computational tasks and algorithms, computability theory has set the stage for the study of the computational resources (like time) required by such algorithms. When this study focuses on the resources that are necessary for any algorithm that solves a particular task (or a task of a particular type), it is viewed as belonging to the theory of computational complexity (also known as "complexity theory"). In contrast, when the focus is on the design and analysis of specific algorithms (rather than on the intrinsic complexity of the task), the study is viewed as belonging to a related area that may be called "algorithmic design and analysis." Furthermore, algorithmic design and analysis tends to be subdivided according to the domain of mathematics, science, and engineering in which the computational tasks arise. In contrast, complexity theory typically maintains a unity of the study of computational tasks that are solvable within certain resources (regardless of the origins of these tasks).

Complexity theory is a central field of the theoretical foundations of computer science. It is concerned with the study of the intrinsic complexity of computational tasks. That is, a typical complexity theoretic study refers to the computational resources required to solve a computational task (or a class of such tasks), rather than referring to a specific algorithm or an algorithmic schema. Actually, research in complexity theory tends to start with and focuses on the computational resources themselves, and addresses the effect of limiting these resources on the class of tasks that can be solved. Thus, computational complexity is the general study of what can be achieved within limited time (and/or other limitations on natural computational resources).

ABSOLUTE GOALS AND RELATIVE RESULTS

Saying that complexity theory is concerned with the study of the intrinsic complexity of computational tasks means that its "final" goal is the determination of the complexity of any well-defined task. An additional goal would be obtaining an understanding of the relations between various computational phenomena (e.g., relat-

Absolute Results (a.k.a. Lower-Bounds)

As stated in the body of this article, absolute results are not known for many of the "big questions" of complexity theory (most notably the P versus NP question). However, several highly non-trivial absolute results have been proved. For example, it was shown that using negation can speed-up the computation of monotone functions (which do not require negation for their mere computation). In addition, many promising techniques were introduced and employed with the aim of providing a low-level analysis of the progress of computation. The interested reader is referred to Arora and Barak's textbook *Complexity Theory: A Modern Approach* [Cambridge University Press, 2009].

ing one fact regarding computational complexity to another). Indeed, we may say that the former is concerned with absolute answers regarding specific computational phenomena, whereas the latter is concerned with questions regarding the relation between computational phenomena.

Interestingly, so far complexity theory has been more successful in coping with goals of the latter ("relative") type. In fact, the failure to resolve questions of the "absolute" type led to the flourishing of methods for coping with questions of the "relative" type. Let us say that, in general, the difficulty of obtaining absolute answers may naturally lead to seeking conditional answers, which may in turn reveal interesting relations between phenomena. Furthermore, the lack of absolute understanding of individual phenomena seems to facilitate the development of methods for relating different phenomena. Anyhow, this is what happened in complexity theory.

Putting aside for a moment the frustration caused by the failure of obtaining absolute answers, there is something fascinating in the success to relate different phenomena: In some sense, relations between phenomena are more revealing than absolute statements about individual phenomena. Indeed, the first example that comes to mind is the theory of NP-completeness. Let us consider this theory, for a moment, from the perspective of these two types of goals.

P, NP, AND NP-COMPLETENESS

Complexity theory has failed to determine the intrinsic complexity of tasks such as finding a satisfying assignment to a given (satisfiable) propositional formula or finding a 3-coloring of a given (3-colorable) graph. But it has succeeded in establishing that these two seemingly different computational tasks are in some sense the same (or, more precisely, are computationally equivalent). This success is amazing and exciting; hopefully the reader shares these feelings. The same feeling of wonder and excitement is generated by many of the other discoveries of complexity theory. Indeed, the reader is invited to join a fast tour of some of the other questions and answers that make up the field of complexity theory.

Complexity theory is concerned with the study of the intrinsic complexity of computational tasks means that its "final" goal is the determination of the complexity of any well-defined task.

We will start with the P versus NP question. Our daily experience is that it is harder to solve a problem than it is to check the correctness of a solution (e.g., think of either a puzzle or a homework assignment). Is this experience merely a coincidence or does it represent a fundamental fact of life (i.e., a property of the world)? Could you imagine a world in which solving any problem is not significantly harder than checking a solution to it? Would the term "solving a problem" not lose its meaning in such a hypothetical (and impossible in our opinion) world? The denial of the plausibility of such a hypothetical world (in which "solving" is not harder than "checking") is what "P different from NP" actually means, where P represents tasks that are efficiently solvable and NP represents tasks for which solutions can be efficiently checked.

The mathematically (or theoretically) inclined reader may also consider the task of proving theorems versus the task of verifying the validity of proofs. Indeed, finding proofs is a special type of the aforementioned task of "solving a problem" (and verifying the validity of proofs is a corresponding case of checking correctness). Again, "P different from NP" means that there are theorems that are harder to prove than to be convinced of their correctness when presented with a proof. This means that the notion of a "proof" is meaningful; that is, proofs do help when seeking to be convinced of the correctness of assertions. Here NP represents sets of assertions that can be efficiently verified with the help of adequate proofs, and P represents sets of assertions that can be efficiently verified from scratch (i.e., without proofs).

In light of the foregoing discussion, it is clear that the P versus NP question is a fundamental scientific question with far-reaching consequences. The fact that this question seems beyond our current reach led to the development of the theory of NP-completeness. Loosely speaking, this theory identifies a set of computational problems that are as hard as NP. That is, the fate of the P versus NP question lies with each of these problems: If any of these problems is easy to solve then so are all problems in NP. Thus, showing that a problem is NP-complete provides evidence to its intractability (assuming, of course, P different than NP). Indeed, demonstrating the NP-completeness of computational tasks is a central tool in indicating hardness of natural computational problems, and it has been used extensively both in computer science and in other disciplines. NP-completeness indicates not only the conjectured intractability of a problem but rather also its "richness," in the sense that the problem is rich enough to encode any other problem in NP. The use of the term "encoding" is justified by the exact meaning of NP-completeness, which in turn establishes relations between different computational problems (without referring to their absolute complexity).

SOME ADVANCED TOPICS

The foregoing discussion of NP-completeness hints to the importance of representation, since it referred to different problems that encode one another. Indeed, the importance of representation is a central aspect of complexity theory. In general, complexity theory is concerned with problems for which the solutions are implicit in the problem's statement (or rather in the instance). That is, the problem (or rather its instance) contains all necessary information, and one merely needs to process this information in order to supply the answer [1]. Thus, complexity theory is concerned with manipulation of information, and its transformation

from one representation (in which the information is given) to another representation (which is the one desired). Indeed, a solution to a computational problem is merely a different representation of the information given; that is, a representation in which the answer is explicit rather than implicit. For example, the answer to the question of whether or not a given Boolean formula is satisfiable is implicit in the formula itself (but the task is to make the answer explicit). Thus, complexity theory clarifies a central issue regarding representation; that is, the distinction between what is explicit and what is implicit in a representation. Furthermore, it even suggests a quantification of the level of non-explicitness.

In general, complexity theory provides new viewpoints on various phenomena that were considered also by past thinkers. Examples include the aforementioned concepts of solutions, proofs, and representation as well as concepts like randomness, knowledge, interaction, secrecy, and learning. We next discuss the latter concepts and the perspective offered by complexity theory.

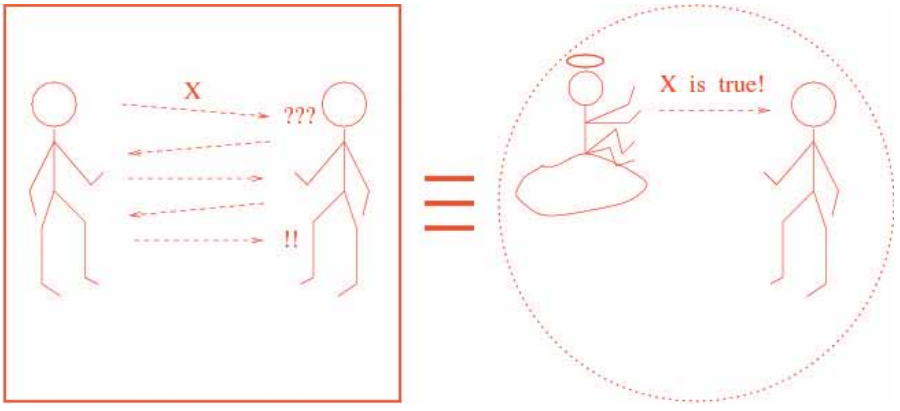
Randomness. The concept of randomness has puzzled thinkers for ages. Their perspective can be described as ontological: They asked "what is randomness" and wondered whether it exists at all (or is the world deterministic). The perspective of complexity theory is behavioristic: It is based on defining objects as equivalent if they cannot be told apart by any efficient procedure. That is, a coin toss is (defined to be) "random" (even if one believes that the universe is deterministic) if it is infeasible to predict the coin's outcome. Likewise, a string (or a distribution on strings) is "random" if it is infeasible to distinguish it from the uniform distribution (regardless of whether or not one can generate the latter). Interestingly, randomness (or rather pseudorandomness) defined this way is efficiently expandable; that is, under a reasonable complexity assumption (to be discussed next), short pseudorandom strings can be deterministically expanded into long pseudorandom strings. Indeed, it turns out that randomness is intimately related to intractability. Firstly, note that the very definition of pseudorandomness refers to intractability (i.e., the infeasibility of distinguishing a pseudorandom object from a uniformly distributed object). Secondly, as stated, a complexity assumption, which refers to the existence of functions that are easy to evaluate but hard to invert (called "one-way functions"), implies the existence of deterministic programs (or "pseudorandom generators") that stretch short, random seeds into long pseudorandom sequences. In fact, it turns out that the existence of pseudorandom generators is equivalent to the existence of one-way functions.

Knowledge. Complexity theory offers its own perspective on the concept of knowledge (and distinguishes it from information). Specifically, complexity theory views knowledge as the result of a hard computation. Thus, whatever can be efficiently done by anyone is not considered knowledge. In particular, the result of an easy computation applied to publicly available information is not considered knowledge. In contrast, the value of a hard-to-compute function applied to publicly available information is knowledge, and if somebody provides you with such a value then it has provided you with knowledge. This discussion is related to the notion of zero-knowledge interactions, which are interactions in which no knowledge is gained (see Figure 1). Such interactions may still be useful, because they may convince a party of the correctness of specific data that was provided beforehand. For example, a zero-knowledge interactive proof may convince a party that a given graph is 3-colorable without yielding any 3-coloring.

Interaction. The foregoing paragraph has explicitly referred to interaction, viewing it as a vehicle for gaining knowledge and/or gaining confidence. Let us highlight the latter application by noting that it may be easier to verify an assertion when allowed to interact with a prover rather than when reading a proof. Put differently, interaction with a good teacher may be more beneficial than reading any book. The added power of such interactive proofs is rooted in their being randomized (i.e., the verification procedure is randomized), because if the verifier's questions can be determined beforehand then the prover may just provide the transcript of the interaction as a traditional written proof.

Secrecy. Another concept related to knowledge is that of secrecy. Knowledge is something that one party may have while another party does not have (and cannot feasibly obtain by itself)—thus, in some sense knowledge is a secret. In general, complexity theory is related to cryptography, where the latter is broadly defined as the study of systems that are easy to use but hard to abuse. Typically, such systems involve secrets, randomness, and interaction as well as a complexity gap between the ease of proper usage and the infeasibility of causing the system to deviate from its prescribed behavior. Thus, much of cryptography is based on complexity-theoretic assumptions and its results are typically transformations of relatively simple computational primi-

Figure 1. An illustration of the concept of zero-knowledge proof.



On the Use of Complexity In Cryptography

A computational complexity gap, captured in the definition of one-way functions, is a necessary and sufficient condition for much of modern cryptography. Loosely speaking, one-way functions are functions that are easy to compute but hard to invert (in an average-case sense). The existence of one-way functions implies that P is different from NP, which means that such a complexity gap is only widely conjectured to exist (rather than known for a fact). We demonstrate the use of this gap in the case of the archetypical cryptographic task of providing secret communication, which in turn is reduced to the construction of encryption schemes.

Encryption schemes are supposed to provide secret communication between parties in a setting in which these communications may be eavesdropped by an adversary. There are two cases differing according to whether or not the communicating parties have agreed on a common secret prior to the communication. In both cases, the encryption scheme consists of three efficient procedures: key generation, encryption (denoted by E), and decryption (D). Loosely speaking, on input a security parameter n, the key-generation procedure outputs a (random) pair of corresponding (n-bit long) encryption and decryption keys, (e, d), such that for every bit string x, it holds that $D_d(E_e(x)) = x$, where $E_e(x)$ (resp., $D_d(y)$) denotes the output of the encryption (resp., decryption) procedure on input (e, x) (resp., (d, y)).

The difference between the two cases lies in the way in which the scheme is employed and this will be reflected in the definition of security. In the first case, known as the private-key case, a set of mutually trustful parties jointly employ the key-generation process, prior to the actual communication, obtaining a pair of keys (e, d). We stress, in this case, the encryption key e is known to all trusted parties and to them only. Later, each trusted party may encrypt messages by applying E_e, and retrieve them (i.e., decrypt) by applying D_d. The information available to the adversary, in this case, is a sequence of encrypted messages, sent over the channel, using a fixed encryption key unknown to it. (The total amount of information encrypted using this encryption key may be much greater than the length of the key, and so perfect information theoretic secrecy is not possible).

In the second case, known as the public-key case, the receiver invokes the key-generation process, publicizes the encryption key e (but not the decryption key d), and the sender uses e to generate encryptions as before. This allows everybody (not only parties that the receiver trusts) to send encrypted messages to the receiver; however, in such a case the adversary also knows the encryption key e. Thus, the information available to the adversary in this case is a sequence of encrypted messages, sent over the channel, using a fixed encryption key that is also known. In both cases, security amounts to asserting that it is infeasible for the adversary to learn anything from the information available to it. That is, whatever the adversary can efficiently compute from the public information can be efficiently computed from scratch.

Note that in the private-key case, we may assume, without loss of generality, that $e = d$; whereas in the public-key case, d must be hard to compute from e. Private-key encryption schemes exist if and only if one-way functions exist. Public-key encryption schemes can be constructed based on a seemingly stronger assumption, yet this assumption is implied by widely believed conjectures such as the conjectured intractability of factoring integers.

BEYOND ENCRYPTION SCHEMES

Cryptography encompasses much more than methods for providing secret communication. Another basic cryptographic task is that of providing authenticated communication, which in turn is reduced to the construction of signature (and/or message authentication) schemes. In general, cryptography is concerned with the construction of schemes that maintain any desired functionality under malicious attempts aimed at making these schemes deviate from their prescribed functionality. Loosely speaking, a secure implementation of a multi-party functionality is a multi-party protocol in which the impact of malicious parties is effectively restricted to application of the prescribed functionality to inputs chosen by the corresponding parties. One major result in this area states that, under plausible assumptions regarding computational difficulty, any efficiently computed functionality can be securely implemented.

tives (e.g., one-way functions) into more complex cryptographic applications (e.g., secure encryption schemes). (See Sidebar "On the Use of Complexity in Cryptography.")

Learning. We have already mentioned the concept of learning when referring to learning from a teacher versus learning from a book. Recall that complexity theory provides evidence to the advantage of the former. This is in the context of gaining knowledge about publicly available information. In contrast, computational learning theory is concerned with learning objects that are only partially available to the learner (i.e., reconstructing a function based on its value at a few random locations or even at locations chosen by the learner). Still, complexity theory sheds light on the intrinsic limitations of learning (in this sense).

Other computational tasks. Complexity theory deals with a variety of computational tasks. We have already mentioned two fundamental types of tasks: searching for solutions (or rather finding solutions) and making decisions (e.g., regarding the validity of assertions). We have also hinted that in some cases these two types of tasks can be related. Now consider two additional types of tasks: counting the number of solutions and generating random solutions. Clearly, both are at least as hard as finding arbitrary solutions to the corresponding problem, but it turns out that for some natural problems they are not significantly harder. Specifically, under some natural conditions on the problem, approximately counting the number of solutions and generating an approximately random solution is not significantly harder than finding an arbitrary solution.

Approximation. Having mentioned the notion of approximation, the study of the complexity of finding "approximate solutions" is also of natural importance. One type of approximation problems refers to an objective function defined on the set of potential solutions: Rather than finding a solution that attains the optimal value, the approximation task consists of finding a solution that attains an "almost optimal" value, where the notion of almost optimal may be understood in different ways giving rise to different levels of approximation. Interestingly, in many cases, even a very relaxed level of approximation is as difficult to obtain as solving the original (exact) search problem (i.e., finding an approximate solution is as hard as finding an optimal solution). Surprisingly, these hardness of approximation results are related to the study of probabilistically checkable proofs, which are proofs that allow for ultra-fast probabilistic verification. Amazingly, every proof can be efficiently transformed into one that allows for probabilistic verification based on probing a constant number of bits (in the alleged proof). Turning back to approximation problems, in other cases a reasonable level of approximation is easier to achieve than solving the original (exact) search problem.

Average-case complexity. Approximation is a natural relaxation of various computational problems. Another natural relaxation is the study of average-case complexity, where the "average" is taken over some "simple" distributions (representing a model of the problem's instances that may occur in practice). Although it was not stated explicitly, the entire discussion so far has referred to "worst-case" analysis of algorithms. Worst-case complexity is a more robust notion than average-case complexity. For starters, one avoids the controversial question of what are the instances that are "important in practice" and correspondingly the selection of the class of distributions for which average-case analysis is to be conducted. Nevertheless, a relatively robust theory of average-case complexity has been suggested, albeit it is less developed than the theory of worst-case complexity.

Randomness extractors. In view of the central role of randomness in complexity theory (as evident, say, in the study of pseudorandomness, probabilistic proof systems, and cryptography), one may wonder as to whether the randomness needed for the various applications can be obtained in real life. One specific question, which received a lot of attention, is the possibility of "purifying" randomness (or "extracting good randomness from bad sources"). That is, can we use "defective" sources of randomness in order to implement almost perfect sources of randomness. The answer depends, of course, on the model of such defective sources. This study turned out to be related to complexity theory, where the tightest connection is between a certain type of randomness extractor and a certain type of pseudorandom generator.

Space complexity. So far we have focused on the time complexity of computational tasks, while relying on the natural association of efficiency with time. However, time is not the only resource one should care about. Another important resource is space: The amount of (temporary) memory consumed by the computation. The study of space complexity has uncovered several fascinating phenomena, which seem to indicate a fundamental difference between space complexity and time complexity. For example, in the context of space complexity, verifying proofs of validity of assertions (of any specific type) has the same complexity as verifying proofs of invalidity for the same type of assertions.

SUMMARY

In case the reader feels dizzy, it is no wonder. We took an ultra-fast aerial tour of some mountain tops, and dizziness is to be expected. More leisurely paced touring experiences are probably offered in courses given by your university.

Further Reading

The P versus NP question and NP-completeness are covered in many basic textbooks. I've written about this subject in *P, NP, and NP-Completeness: The Basics of Complexity Theory*, published by Cambridge University Press. If you are interested in advanced topics, *Computational Complexity: A conceptual perspective* (Cambridge University Press, 2008) is more comprehensive.

Biography

Oded Goldreich is a professor of computer science at the Weizmann Institute of Science, Israel. He is an active researcher with numerous contributions to complexity theory, cryptography and related areas in the theory of computation. He has published a number of books in these domains, including *Foundations of Cryptography, Vol 1: Basic Tools* (2001) and *Vol 2: Basic Applications* (2004), Cambridge University Press.

Note

[1] In contrast, in other disciplines, solving a problem may require gathering information that is not available in the problem's statement. This information may either be available from auxiliary (past) records or be obtained by conducting new experiments.

© 2012 ACM 1528-4972/12/03 $10.00