A brief overview of Complexity Theory

The following brief overview is intended as a teaser for students in an introductory course on Complexity Theory.

Out of the tough came forth sweetness[1]

Judges, 14:14

Complexity Theory is concerned with the study of the *intrinsic complexity* of computational tasks. Its "final" goals include the determination of the complexity of any well-defined task. Additional "final" goals include obtaining an understanding of the relations between various computational phenomena (e.g., relating one fact regarding computational complexity to another). Indeed, we may say that the former type of goals is concerned with *absolute* answers regarding specific computational phenomena, whereas the latter type is concerned with questions regarding the *relation* between computational phenomena.

Interestingly, the current success of Complexity Theory in coping with the latter type of goals has been more significant. 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. Putting aside for a moment the frustration caused by the failure, we must admit that there is something fascinating in the success: in some sense, establishing relations between phenomena is more revealing than making statements about each phenomenon. 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.

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 established that these two seemingly different computational tasks are in some sense the same (or, more precisely, are computationally equivalent). The author finds this success amazing and exciting, and hopes that the reader shares his feeling. 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.

We will indeed 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 research problem). Is this experience merely a coincidence or does it represent a fundamental fact of life (or 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

[0]Written by Oded Goldreich (2005).

[1]The quote is commonly used to mean that benefit arose out of misfortune.

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 (i.e., that proofs do help when trying 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 of 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 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 is based on establishing relations between different computational problems (without referring to their "absolute" complexity).

The foregoing discussion of the P-versus-NP Question also hints to *the importance of representation*, a phenomenon that is central to complexity theory. In general, complexity theory is concerned with problems the solutions of which are implicit in the problem's statement. That is, the problem contains all necessary information, and one merely needs to process this information in order to supply the answer.[2] 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 rep-

[2]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.

resentation; 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 proofs and representation as well as concepts like randomness, knowledge, interaction, secrecy and learning. We next discuss some of these concepts and the perspective offered by complexity theory.

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 exist 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 of 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 pseudorandomness object from a uniformly distributed object). Secondly, as hinted above, a complexity assumption that refers to the existence of functions that are easy to evaluate but hard to invert (called *one-way functions*) imply the existence of deterministic programs (called *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.

Complexity theory offers its own perspective on the concept of *knowledge* (and distinguishes it from information). It 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. Such interactions may still be useful, because they may assert the *correctness* of specific data that was provided beforehand.

The foregoing paragraph has explicitly referred to *interaction*. It has pointed one possible motivation for interaction: gaining knowledge. It turns out that interaction may help in a variety of other contexts. For example, 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 some teacher may be more beneficial than reading any book. We comment that 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.

Another concept related to knowledge is that of *secrecy*: knowledge is something that one party has 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 primitives (e.g., one-way functions) into more complex cryptographic applications (e.g., a secure encryption scheme).

We have already mentioned the context 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., learning a function based on its value at a few random locations or even at locations chosen by the learner). Complexity theory sheds light on the intrinsic limitations of learning (in this sense).

Complexity theory deals with a variety of computational tasks. We have already mentioned two fundamental types of tasks: *searching for solutions* (or "finding solutions") and *making decisions* (e.g., regarding the validity of assertion). We have also hinted that in some cases these two types of tasks can be related. Now we consider two additional types of tasks: *counting the number of solutions* and *generating random solutions*. Clearly, both the latter tasks 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.

Having mentioned the notion of *approximation*, we note that the study of the complexity of finding approximate solutions has also received a lot of attention. 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 achieve as 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, we note that in other cases a reasonable level of approximation is easier to achieve than solving the original (exact) search problem.

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). We stress that, although it was not stated explicitly, the entire discussion so far has referred to "worst-case" analysis of algorithms. We mention that 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 far less developed than the theory of worst-case complexity.

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 "defected" sources of randomness in order to implement almost perfect sources of randomness. The answer depends, of course, on the model of such defected sources. This study turned out to be related to complexity theory, where the most tight connection is between some type of *randomness extractors* and some type of pseudorandom generators.

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.

In case the reader feels dizzy, it is no wonder. We took an ultra-fast air-tour of some mountain tops, and dizziness is to be expected. Needless to say, the rest of the course will be in a totally different style. We will climb some of these mountains by foot, step by step, and will stop to look around and reflect.

Absolute Results (a.k.a. Lower-Bounds). As stated up-front, 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. However, the focus of this course is elsewhere.