On Teaching the Basics of Complexity Theory*

Oded Goldreich
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, ISRAEL.
Email: oded.goldreich@weizmann.ac.il

July 25, 2005

Abstract

We outline a conceptual framework for teaching the basic notions and results of complexity theory. Our focus is on using definitions and on organizing the presentation in a way that reflects the fundamental nature of the material. We do not attempt to provide a self-contained presentation of the material itself, but rather outline our (non-innovative) suggestions regarding how this material should be presented in class.

We discuss the P-vs-NP Question, the general notion of a reduction, and the theory of NP-completeness. In particular, we suggest presenting the P-vs-NP Question both in terms of search problems and in terms of decision problems (where NP is viewed as a class of proof systems). As for the theory of NP-completeness, we suggest highlighting the mere existence of NP-complete sets.

1 Introduction

Shimon Even had a passion for good teaching, and so writing this essay in his memory seems most appropriate. Good teaching is an art, and Shimon was one of the best masters. It is hard (if at all possible) to cultivate artistic talents, but there are certain basic principles that underly each art form, and these can be discussed.

One central aspect of good teaching is putting things in the right perspective; that is, a perspective that clarifies the motivation for the various definitions and results. It is easy to provide a good perspective on the basic notions and results of complexity theory, because these are of fundamental nature and of great intuitive appeal. Unfortunately, often this is not the way this material is taught. The annoying (and quite amazing) consequences are students that have only a vague understanding of the *meaning* of these fundamental notions and results.

1.1 The source of trouble and eliminating it

In my opinion, it all boils down to taking the time to explicitly discuss the meaning of definitions and results. A related issue is using the "right" definitions (i.e., those that reflect better the fundamental nature of the notion being defined) and teaching things in the (conceptually) "right" order. Two concrete examples follow.

*This essay was written for the current volume. The technical presentation was adapted from earlier lecture notes [4].

Typically, NP is defined as the class of languages recognized by non-deterministic polynomial-time machines. Even bright students may have a hard time figuring out (by themselves) why one should care about such a class. On the other hand, when defining NP as the class of assertions that have easily verifiable proofs, each student is likely to understand its fundamental nature. Furthermore, the message becomes even more clear when discussing the search version analogue.

Similarly, one typically takes the students throughout the detailed proof of Cook's Theorem before communicating to them the striking message (i.e., that "universal" problems exist at all, let alone that many natural problems like SAT are universal). Furthermore, in some cases, this message is not communicated explicitly at all.

1.2 Concrete suggestions

The rest of this essay provides concrete suggestions for teaching the basics of complexity theory. The two most important suggestions were already mentioned above:

1. The teacher should communicate the fundamental nature of the P-vs-NP Question while referring to definitions that (clearly) reflect this nature.

2. The teacher should communicate the striking significance of the mere existence of NP-complete problems (let alone natural ones) before exhausting the students with complicated reductions.

In addition, I suggest setting the stage for the course (or series of lectures) by providing a "definition" of complexity theory. I would say that this is a central field of Theoretical Computer Science, concerned with the study of the *intrinsic* complexity of computational tasks, where this study tend to aim at *generality*: It focuses on natural computational resources, and the effect of limiting these resources on the *class of problems* that can be solved. Put in other words, Complexity Theory aims at understanding the *nature of efficient computation*.

I advocate a model-independent presentation of the results. Indeed, the presentation needs to refer to the specifics of the model of computation only when encoding the relation between consecutive instantaneous configuration of computation (see Section 4.3).

Finally, until we reach the day in which every student can be assumed to have understood the meaning of the P-vs-NP Question and of NP-completeness, I suggest not to assume such an understanding when teaching an advanced complexity theory course. Instead, I suggest starting with a fast discussion of this basic material, making sure that the students understand its conceptual meaning.[1] In fact, this essay is based on my notes [4] for three lectures (covering the basic material), which were given in a graduate course on complexity theory.

1.3 A paranthetical comment on computability versus complexity

This essay refers to the current situation in many schools, where the basics of complexity theory are taught within a course in which material entitled "computability" plays at least an equal role. The essay is confined to the "complexity" part of such a course.

In my opinion, complexity theory should play the main role in a course on computability and complexity theory. The basic concepts and results of computability theory should be regarded as an important preliminary material, which sets the stage for the study of the complexity of the computational tasks that can be automated at all. Thus, the computability aspects of such a

[1] Needless to say, the rest of the course should also clarity the conceptual meaning of the material being taught.

course should be confined to establishing that the intuitive notion of an algorithm can be rigorously defined, and to emphasizing the uncomputability of most functions and of some natural functions (e.g., the Halting predicate). This includes introducing the idea of a universal algorithm, but does not included extensive programming with Turing machines or extensive study of (complexity-free) Turing reductions. Needless to say, I oppose the teaching of finite automata (let alone context-free grammers) within such a course.

Articulating the opinions expresses in the last paragraph is beyond the scope of the current essay. On the other hand, the rest of this essay is independent of the foregoing controversy. That is, it refers to the basic material of complexity theory, independent of the question within which course this material is taught and what role does it play in such a course.

1.4 Contents

This essay is intended for computer scientists who teach or plan to teach a course on computability and complexity theory. It focuses on material that is typically taught in such a course, and is probably well-known to the reader. Thus, *the focus is not on the material itself, but rather on how it should be presented in class.*

In addition, the essay mentions some topics that are typically not covered in a basic course on computability and complexity theory. These topics include self-reducibility (of search problems), the existence of NP-sets that are neither in P nor NP-complete, the effect of having coNP-sets that are NP-complete, the existence of optimal search algorithms for NP-relations, and the notion of promise problems.

Organization: Section 2 contains a presentation of the P-vs-NP Question both in terms of search problems and in terms of decision problems. Section 3 contains a general treatment of reductions as well as a subsection of self-reducibility. Section 4 contains a presentation of the basic definitions and results of the theory of NP-completeness. Section 5 covers three additional topics.

2 P versus NP

Most students have heard of P and NP before, but we suspect that many have not obtained a good explanation of what the P vs NP Question actually represents. This unfortunate situation is due to using the standard technical definition of NP (which refers to non-deterministic polynomial-time) rather than more cumbersome definitions that clearly capture the fundamental nature of NP. Below, we take the alternative approach. In fact, we present two fundamental formulations of the P vs NP Question, one in terms of search problems and the other in terms of decision problems.

Efficient computation. The teacher should discuss the association of efficiency with polynomial-time, stressing that this association merely provides a convenient way of addressing fundamental issues. In particular, polynomials are merely a "closed" set of moderately growing functions, where "closure" means closure under addition, multiplication and functional composition. These closure properties guarantee the closure of the class of efficient algorithm under natural composition of algorithms. (The specifics of the model of computation are also immaterial, as long as the model is "reasonable"; this strengthening of the Church–Turing Thesis is called the Cobham–Edmonds Thesis.)

2.1 The search version: finding versus checking

In the eyes of non-experts, search problems are more natural than decision problems: typically, people seeks solutions more than they stop to wonder whether or not solutions exist. Thus, we recommend to start by discussing the fundamental implication of the P-vs-NP Question on search problems. Admittingly, the cost is more cumbersome formulations, but it is more than worthwhile. Furthermore, the equivalence to the decision problem formulation gives rise to conceptually appealing exercises.

We focus on polynomially-bounded relations, where a relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ is polynomially-bounded if there exists a polynomial p such that for every $(x,y) \in R$ it holds that $|y| \leq p(|x|)$. For such a relation it makes sense to ask whether, given an "instance" x, one can efficiently find a "solution" y such that $(x,y) \in R$. The polynomial bound on the length of the solution (i.e., y) guarantees that the intrinsic complexity of outputting a solution may not be due to the length (or mere typing) of the required solution.

The class P as a natural class of search problems. With each polynomially-bounded relation R, we associate the following search problem: *given x find y such that $(x,y) \in R$ or state that no such y exists.* The class $\mathcal{P}$ *corresponds* to the class of search problems that are solvable in polynomial-time; that is, a relation R (or rather the search problem of R) is polynomial-time solvable if there exists a polynomial-time algorithm that given x find y such that $(x,y) \in R$ or state that no such y exists.

The class NP as another natural class of search problems. A polynomially-bounded relation R is called an NP-relation if, given an alleged instance-solution pair, one can efficiently check whether or not the pair is valid; that is, there exists a polynomial-time algorithm that given x and y determines whether or not $(x,y) \in R$. It is reasonable to focus on search problems for NP-relations, because the ability to efficiently recognize a valid solution seems to be a natural prerequisite for a discussion regarding the complexity of finding such solutions. (Indeed, one can introduce (unnatural) non-NP-relations for which the search problem is solvable in polynomial-time; still the restriction to NP-relations is very natural.)

The P versus NP question in terms of search problems: *Is it the case that the search problem of any NP-relation can be solved in polynomial-time?* In other words, if it is easy to check whether or not a given solution for a given instance is correct then is it also easy to find a solution to a given instance?

If $\mathcal{P} = \mathcal{NP}$ then this would mean that if solutions to given instances can be efficiently verified for correctness then they can also be efficiently found (when given only the instance). This would mean that all reasonable search problems (i.e., all NP-relations) are easy to solve. Needless to say, such a situation would contradict the intuitive feeling that some reasonable search problems are hard to solve. On the other hand, if $\mathcal{P} \neq \mathcal{NP}$ then there exist reasonable search problems (i.e., some NP-relations) that are hard to solve. This conforms with our basic intuition by which some reasonable problems are easy to solve whereas others are hard to solve.

2.2 The decision version: proving versus verifying

We suggest starting by asserting the natural stature of decision problems (beyond their role in the study of search problems). After all, some people do care about the truth, and so determining whether a given object has some claimed property is an appealing problem. The P-vs-NP Question

Recall that search problems refer to binary relations. For such a relation R, the corresponding search problem is given x to find y such that $(x,y) \in R$ (or assert that no such y exists). We suggest defining two classes of search problems.

- $\mathcal{PF}$ (standing for "Poly-Find") denotes the class of search problems that are solvable in polynomial-time. That is, $R \in \mathcal{PF}$ if there exists a polynomial time algorithm that given x finds y such that $(x,y) \in R$ (or assert that no such y exists).
- $\mathcal{PC}$ (standing for "Poly-Check") denotes the class of search problems that correspond to polynomially-bounded binary relations that are "checkable" in polynomial-time. That is, $R \in \mathcal{PC}$ if the following two conditions hold
  1. For some polynomial p, if $(x,y) \in R$ then $|y| \le p(|x|)$.
  2. There exists a polynomial-time algorithm that given (x,y) determines whether or not $(x,y) \in R$.
- In terms of search problems the P-vs-NP Question consists of asking whether or not $\mathcal{PC}$ is contained in $\mathcal{PF}$. The conjectured inequality $\mathcal{P} \neq \mathcal{NP}$ implies that $\mathcal{PC} \setminus \mathcal{PF} \neq \emptyset$.

Figure 1: P-vs-NP in terms of search problems: notational suggestions.

refers to the complexity of answering such questions for a wide and natural class of properties associated with the class $\mathcal{NP}$. The latter class refers to properties that have efficient proof systems allowing for the verification of the claim that a given object has a predetermined property (i.e., is a member of a predetermined set).

For an NP-relation R, we denote the set of instances having a solution by L_R; that is, $L_R = \{x : \exists y\ (x,y) \in R\}$. Such a set is called an NP-set, and $\mathcal{NP}$ denotes the class of all NP-sets. Intuitively, an NP-set is a set of valid statements (i.e., statements of membership of a given x in L_R) that can be efficiently verified when given adequate proofs (i.e., a corresponding NP-witness y such that $(x,y) \in R$). This leads to viewing NP-sets as proof systems.

NP-proof systems. Proof systems are defined in terms of their verification procedures. Here we focus on the natural class of efficient verification procedures, where efficiency is represented by polynomial-time computations. (We should either require that the time is polynomial in terms of the statement or confine ourselves to "short proofs" – that is, proofs of length that is bounded by a polynomial in the length of the statement.) NP-relations correspond to proof systems with efficient verification procedures. Specifically, the NP-relation R corresponds to the (proof system with a) verification procedure that checks whether or not the alleged statement-proof pair is in R. This proof system satisfies the natural completeness and soundness conditions: every true statement (i.e., $x \in L_R$) has a valid proof (i.e., an NP-witness y such that $(x,y) \in R$), whereas false statements (i.e., $x \notin L_R$) have no valid proofs (i.e., $(x,y) \notin R$ for all y's).

The P versus NP question in terms of decision problems: *Is it the case that NP-proofs are useless?* That is, is it the case that for every efficiently verifiable proof system one can easily determine the validity of assertions (without being given suitable proofs). If that were the case, then proofs would be meaningless, because they would have no fundamental advantage over directly determining the validity of the assertion. Denoting by $\mathcal{P}$ the class of sets that can be decided efficiently (i.e., by a polynomial-time algorithm), the conjecture $\mathcal{P} \neq \mathcal{NP}$ asserts that proofs are useful: there exists NP-sets that cannot be decided by a polynomial-time algorithm, and so for these sets obtaining a proof of membership (for some instances) is useful (because we cannot efficiently determine membership by ourselves).

Recall that decision problems refer to membership in sets. We suggest defining two classes of decision problems, which indeed coincide with the standard definitions of $\mathcal{P}$ and $\mathcal{NP}$.

- $\mathcal{P}$ denotes the class of decision problems that are solvable in polynomial-time. That is, $S \in \mathcal{P}$ if there exists a polynomial time algorithm that given x determines whether or not $x \in S$.
- $\mathcal{NP}$ denotes the class of decision problems that have NP-proof systems. The latter are defined in terms of a (deterministic) polynomial-time verification algorithm. That is, $S \in \mathcal{NP}$ if there exists a polynomial p and a polynomial-time algorithm V that satisfy the following completeness and soundness conditions:

  1. Completeness: if $x \in S$ then there exists y of length at most $p(|x|)$ such that $V(x,y) = 1$. (Such a string y is called an NP-witness for $x \in S$.)
  2. Soundness: if $x \notin S$ then for every y it holds that $V(x,y) = 0$.

  Indeed, the point is defining $\mathcal{NP}$ as a class of sets of assertions having efficient verification procedures.
- In terms of decision problems the P-vs-NP Question consists of asking whether or not $\mathcal{NP}$ is contained in $\mathcal{P}$. Since $\mathcal{P} \subseteq \mathcal{NP}$, the question is phrase as whether or not $\mathcal{NP}$ equals $\mathcal{P}$.

Figure 2: P-vs-NP in terms of decision problems: notational suggestions.

2.3 Equivalence of the two formulations

We strongly recommend proving in class that *the two formulations of the P-vs-NP Questions are equivalent.* That is, the search problem of every NP-relation is solvable in polynomial time if and only if membership in any NP-set can be decided in polynomial time.[2] This justifies the focus on the latter (simpler) formulation.

We also suggest mentioning that $\mathcal{NP}$ is sometimes defined as the class of sets that can be decided by a *fictitious* device called a non-deterministic polynomial-time machine (and that this is the source of the notation NP). The reason that this class of fictitious devices is important is because it captures (indirectly) the definition of NP-proofs. We suggest proving that indeed the definition of $\mathcal{NP}$ in terms of non-deterministic polynomial-time machine equals our definition of $\mathcal{NP}$ (in terms of the class of sets having NP-proofs).

[2] Using the notations of Figure 1, we prove that $\mathcal{PC} \subseteq \mathcal{PF}$ if and only if $\mathcal{NP} = \mathcal{P}$. Suppose, on the one hand, that the inclusion holds for the search version (i.e., $\mathcal{PC} \subseteq \mathcal{PF}$). Let L be an arbitrary NP-set and R_L be the corresponding witness relation. Then R_L is a NP-relation, and by the hypothesis its search problem is solvable in polynomial time (i.e., $R_L \in \mathcal{PC} \subseteq \mathcal{PF}$). This yields a polynomial-time decision procedure for L; i.e., given x try to find y such that $(x,y) \in R_L$ (and output "yes" iff such a y was found). Thus, $\mathcal{NP} = \mathcal{P}$ follows. Suppose, on the other hand, that $\mathcal{NP} = \mathcal{P}$ (as classes of sets), and let R be an arbitrary NP-relation. Then the set $S_R \stackrel{\rm def}{=} \{(x,y') : \exists y'' \text{ s.t. } (x,y'y'') \in R\}$ is in $\mathcal{NP}$ and hence in $\mathcal{P}$. This yields a polynomial-time algorithm for solving the search problem of R, by extending a prefix of a potential solution bit-by-bit (while using the decision procedure to determine whether or not the current prefix is valid). Thus, $\mathcal{PC} \subseteq \mathcal{PF}$ follows.

3 Reductions and Self-reducibility

We assume that all students have heard of reductions, but again we fear that most have obtained a conceptually-poor view of their nature. We present first the general notion of (polynomial-time) reductions among computational problems, and view the notion of a Karp-reduction as an important special case that suffices (and is more convenient) in many cases.

3.1 The general notion of a reduction

Reductions are procedures that use "functionally specified" subroutines. That is, the functionality of the subroutine is specified, but its operation remains unspecified and its running-time is counted at unit cost. Analogously to algorithms, which are modeled by Turing machines, reductions can be modeled as *oracle* (Turing) machines. A reduction solves one computational problem (which may be either a search or decision problem) by using oracle (or subroutine) calls to another computational problem (which again may be either a search or decision problem). We focus on efficient (i.e., polynomial-time) reductions, which are often called Cook reductions.

The key property of reductions is that they translate efficient procedures for the subroutine into efficient procedures for the invoking machine. That is, if one problem is Cook-reducible to another problem and the latter is polynomial-time solvable then so is the former.

The most popular case is of reducing decision problems to decision problems, but we will also consider reducing search problems to search problems or reducing search problems to decision problems. (Indeed, a good exercise is to show that the search problem of any NP-relation can be reduced to deciding membership in some NP-set (see Footnote 2).)

A Karp-reduction is a special case of a reduction (from a decision problem to a decision problem). Specifically, for decision problems L and L', we say that L is Karp-reducible to L' if there is a reduction of L to L' *that operates as follows*: On input x (an instance for L), the reduction computes x', makes query x' to the oracle L' (i.e., invokes the subroutine for L' on input x'), and answers whatever the latter returns. This Karp-reduction is often represented by the polynomial-time computable mapping of x to x'; that is, a polynomial-time computable f is called a Karp-reduction of L to L' if for every x it holds that $x \in L$ iff $f(x) \in L'$.

Indeed, a Karp-reduction is a syntactically restricted notion of a reduction. This restricted case suffices for many cases (e.g., most importantly for the theory of NP-completeness (when developed for decision problems)), but not in case we want to reduce a search problem to a decision problem. Furthermore, whereas each decision problem is reducible to its complement, some decision problems are not Karp-reducible to their complement (e.g., the trivial decision problem).[3] Likewise, each decision problem in $\mathcal{P}$ is reducible to any computational problem by a reduction that does not use the subroutine at all, whereas such a trivial reduction is disallowed by the syntax of Karp-reductions. (Nevertheless, a popular exercise of dubious nature is to show that any decision problem in $\mathcal{P}$ is Karp-reducible to any *non-trivial* decision problem.)

We comment that Karp-reductions may (and should) be augmented in order to handle reductions of search problems to search problems. Such an augmented Karp-reduction of the search problem of R to the search problem of R' operates as follows: On input x (an instance for R), the reduction computes x', makes query x' to the oracle R' (i.e., invokes the subroutine for searching R' on input x') obtaining y' such that $(x', y') \in R'$, and uses y' to compute a solution y to x (i.e., $(x, y) \in R$). Thus, such a reduction can be represented by two polynomial-time computable mappings, f and g, such that $(x, g(x, y')) \in R$ for any y' that solves $f(x)$ (i.e., y' satisfies $(f(x), y') \in R'$).

[3] We call a decision problem trivial if it refers to either the empty set or the set of all strings.

(Indeed, in general, unlike in the case of decision problems, the reduction cannot just return y' as an answer to x.)

We say that two problems are computationally equivalent if they are reducible to one another. This means that the two problems are essentially as hard (or as easy).

3.2 Self-reducibility of search problems

We suggest introducing the notion of self-reducibility[4] for several reasons. Most importantly, it further justifies the focus on decision problems (see discussion following Proposition 1). In addition, it illustrates the general notion of a reduction, and asserts its relevance beyond the theory of NP-completeness.

The search problem of R is called self-reducible if it can be reduced to the decision problem of $L_R = \{x : \exists y\ (x, y) \in R\}$. Note that the decision problem of L_R is always reducible to the search problem for R (e.g., invoke the search subroutine and answer YES if and only if it returns some string (rather than the "no solution" symbol)).

We will see that all NP-relations that correspond to NP-complete sets are self-reducible, mostly via "natural reductions". We start with SAT, the set of satisfiable Boolean formulae (in CNF). Let R_{SAT} be the set of pairs (ϕ, τ) such that τ is a satisfying assignment to the formulae ϕ. Note that R_{SAT} is an NP-relation (i.e., it is polynomially-bounded and easy to decide (by evaluating a Boolean expression)).

Proposition 1 (R_{SAT} is self-reducible): *The search problem of R_{SAT} is reducible to SAT.*

Thus, the search problem of R_{SAT} is computationally equivalent to deciding membership in SAT. Hence, in studying the complexity of SAT, we also address the complexity of the search problem of R_{SAT}. This justifies the relevance of decision problems to search problems in a stronger sense than established in Section 2.3: The study of decision problems determines not only the complexity of the class of "NP-search" problems but rather determines the complexity of each individual search problem that is self-reducible.

Proof: Given a formula ϕ, we use a subroutine for SAT in order to find a satisfying assignment to ϕ (in case such an assignment exists). First, we query SAT on ϕ itself, and return "no solution" if the answer we get is 'false'. Otherwise, we let τ, initiated to the empty string, denote a prefix of a satisfying assignment of ϕ. We proceed in iterations, where in each iteration we extend τ by one bit. This is done as follows: First we derive a formula, denoted ϕ', by setting the first $|\tau| + 1$ variables of ϕ according to the values $\tau 0$. Next we query SAT on ϕ' (which means that we ask whether or not $\tau 0$ is a prefix of a satisfying assignment of ϕ). If the answer is positive then we set $\tau \leftarrow \tau 0$ else we set $\tau \leftarrow \tau 1$ (because if τ is a prefix of a satisfying assignment of ϕ and $\tau 0$ is not a prefix of a satisfying assignment of ϕ then $\tau 1$ must be a prefix of a satisfying assignment of ϕ).

A key point is that each formula ϕ' (which contains Boolean variables as well as constants) can be simplified to contain no constants (in order to fit the canonical definition of SAT, which disallows Boolean constants). That is, after replacing some variables by constants, we should simplify clauses according to the straightforward boolean rules (e.g., a false literal can be omitted from a clause and a true literal appearing in a clause yields omitting the entire clause). ■

[4]Our usage of this term differs from the traditional one. Traditionally, a decision problem is called self-reducible if it is Cook-reducible to itself via a reduction that on input x only makes queries that are smaller than x (according to some appropriate measure on the size of strings). Under some natural restrictions (i.e., the reduction takes the disjunction of the oracle answers) such reductions yield reductions of search to decision (as discussed in the main text).

Advanced comment: A reduction analogous to the one used in the proof of Proposition 1 can be presented also for other NP-search problems (and not only for NP-complete ones).[5] Consider, for example, the problem Graph 3-Colorability and prefixes of a 3-coloring of the input graph. Note, however, that in this case the process of getting rid of constants (representing partial solutions) is more involved. Details are left as an exercise.[6] In general, if you don't see a "natural" self-reducibility process for some NP-complete relation, you should know that a self-reduction process does exist (alas it maybe not be a natural one).

Theorem 2 *The search problem of the NP-relation of any NP-complete set is self-reducible.*

Proof: Let R be an NP-relation of the NP-complete set L_R. In order to reduce the search problem of R to deciding L_R, we compose the following three reductions:

1. The search problem of R is reducible to the search problem of R_{SAT} (by the NP-completeness of the latter).

2. The search problem of R_{SAT} is reducible to SAT (by Proposition 1).

3. The decision problem SAT is reducible to the decision problem L_R (by the NP-completeness of the latter).

The theorem follows. ■

4 NP-completeness

Some (or most) students heard of NP-completeness before, but we suspect that many have missed important conceptual points. Specifically, we stress that the mere existence of NP-complete sets (regardless of whether this is SAT or some other set) is amazing.

4.1 Definitions

The standard definition is that a set is NP-complete if it is in $\mathcal{NP}$ and every set in $\mathcal{NP}$ is reducible to it via a Karp-reduction. Indeed, there is no reason to insist on Karp-reductions (rather than using arbitrary reductions), except that the restricted notion suffices for all positive results and is easier to work with.

We will also refer to the search version of NP-completeness. We say that a binary relation is NP-complete if it is an NP-relation and every NP-relation is reducible to it.

We stress that the mere fact that we have defined something (i.e., NP-completeness) does not mean that this thing exists (i.e., that there exist objects that satisfy the property). *It is indeed remarkable that NP-complete problems do exist.* Such problems are "universal" in the sense that solving them allows to solve any other (reasonable) problem.

[5]We assume that the students have heard of NP-completeness. If this assumption does not hold for your class, then the presentation of the following material should be postponed (to Section 4.1 or to an even later stage).

[6]Hint: At each iteration we wish to determine the relation between the color of the current vertex and the colors of the vertices determined so far. We may test equality (and inequality) between the colors that may be assigned to two vertices by using adequate gadgets, which are connected to the end-points of the vertices we wish to test.

4.2 The existence of NP-complete problems

We suggest not to confuse the mere existence of NP-complete problems, which is remarkable by itself, with the even more remarkable existence of "natural" NP-complete problems. We believe that the following proof allows to deliver this message as well as to focus on the essence of NP-completeness, rather than on more complicated technical details.

Theorem 3 *There exist NP-complete relations and sets.*

Proof: The proof (as well as any other NP-completeness proof) is based on the observation that some NP-relations (resp., NP-sets) are "rich enough" to encode all NP-relations (resp., NP-sets). This is most obvious for the "generic" NP-relation, denoted R_U (and defined below), which is used to derive the simplest proof of the current theorem.

The relation R_U consists of pairs $(\langle M, x, 1^t \rangle, y)$ such that M is a description of a (deterministic) Turing machine that accepts the pair (x, y) within t steps, where $|y| \leq t$. (Instead of requiring that $|y| \leq t$, one may require that M is canonical in the sense that it reads its entire input before halting.) It is easy to see that R_U is an NP-relation, and thus $L_U \stackrel{\mathrm{def}}{=} \{X : \exists y\ (X, y) \in R_U\}$ is an NP-set. Indeed, R_U is recognizable by a universal Turing machine, which on input $(\langle M, x, 1^t \rangle, y)$ emulates (t steps of) the computation of M on (x, y), and U indeed stands for universal (machine). (Thus, the proof extends to any reasonable model of computation, which has adequate universal machines.)

We now turn to showing that any NP-relation is reducible to R_U. As a warm-up, let us first show that any NP-set is Karp-reducible to L_U. Let R be an NP-relation, and $L_R = \{x : \exists y\ (x, y) \in R\}$ be the corresponding NP-set. Let p_R be a polynomial bounding the length of solutions in R (i.e., $|y| \leq p_R(|x|)$ for every $(x, y) \in R$), let M_R be a polynomial-time machine deciding membership (of alleged (x, y) pairs) in R, and let t_R be a polynomial bounding its running-time. Then, the Karp-reduction maps an instance x (for L) to the instance $\langle M_R, x, 1^{t_R(|x|+p_R(|y|))} \rangle$.

Note that this mapping can be computed in polynomial-time, and that $x \in L$ if and only if $\langle M_R, x, 1^{t_R(|x|+p_R(|y|))} \rangle \in L_U$.

To reduce the search problem of R to the search problem of R_U, we use essentially the same reduction. On input an instance x (for R), we make the query $\langle M_R, x, 1^{t_R(|x|+p_R(|y|))} \rangle$ to the search problem of R_U and return whatever the latter returns. Note that if $x \notin L_R$ then the answer will be "no solution", whereas for every x and y it holds that $(x, y) \in R$ if and only if $(\langle M_R, x, 1^{t_R(|x|+p_R(|y|))} \rangle, y) \in R_U$. ■

Advanced comment. Note that the role of 1^t in the definition of R_U is to make R_U an NP-relation. In contrast, consider the relation $R_H \stackrel{\mathrm{def}}{=} \{(\langle M, x \rangle, y) : M(xy) = 1\}$ (which corresponds to the halting problem). Indeed, the search problem of any relation (an in particular of any NP-relation) is Karp-reducible to the search problem of R_H, but the latter is not solvable at all (i.e., there exists no algorithm that halts on every input and on input X outputs y such that $(x, y) \in R_H$ iff such a y exists).

4.3 CSAT and SAT

We suggest establishing the NP-completeness of SAT by a reduction from the circuit satisfaction problem (CSAT), after establishing the NP-completeness of the latter. Doing so allows to decouple two technical issues in the proof of the NP-completeness of SAT: the emulation of Turing machines by circuits, and the encoding of circuits by formulae with auxiliary variables. Following is a rough outline, which focuses on the decision version.

CSAT. Define Boolean circuits as directed acyclic graphs with internal vertices, called gates, labeled by Boolean operations (of arity either 2 or 1), and external vertices called terminals that are associated with either inputs or outputs. When setting the inputs of such a circuit, all internal nodes are assigned values in the natural way, and this yields a value to the output(s), called an evaluation of the circuit on the given input. Define the satisfiability problem of such circuits as determining, for a given circuit, whether there exists a setting to its inputs that makes its (first) output evaluate to 1. Prove the NP-completeness of the circuit satisfaction problem (CSAT), by reducing any NP-set to it (where the set is represented by the machine that recognizes the corresponding NP-relation). The reduction boils down to encoding possible computations of a Turing machine by a corresponding layered circuit, where each layer represents an instantaneous configuration of the machine, and the relation between consecutive configurations is captured by ("uniform") local gadgets in the circuit. (The proof extends to any other "reasonable" model of efficient computation.)

Specifically, for a machine M_R (as in the proof of Theorem 3), we represent the computation of M_R on input (x, y), where x is the input to the reduction and y is undetermined, by a circuit C_x that takes such a string y as input. That is, $C_x(y) = 1$ if and only if M_R accepts (x, y), and so C_x is satisfiable if and only if $x \in L_R$. The circuit C_x consists of layers such that the i^{th} layers of wires (connecting the $i - 1^{\mathrm{st}}$ and i^{th} layers of vertices) represents the instantaneous configuration of $M_R(x, y)$ just before the i^{th} step. In particular, the gates of the $i + 1^{\mathrm{st}}$ layer are designed to guaranteed that the instantaneous configuration of $M_R(x, y)$ just before the i^{th} step is transformed to the instantaneous configuration of $M_R(x, y)$ just before the $i + 1^{\mathrm{st}}$ step.

The above reduction is called "generic" because it (explicitly) refers to any (generic) NP-set. However, the common practice is to establish NP-completeness by a reduction from some NP-complete set (i.e., a set already shown to be NP-complete). This practice is based on the fact that if an NP-complete problem Π is reducible to some problem Π' in NP then Π' is NP-complete. The proof of this fact boils down to asserting the transitivity of reductions.

SAT. Define Boolean formulae, which may be viewed as Boolean circuits with a tree structure. Prove the NP-completeness of the formula satisfaction problem (SAT), even when the formula is given in a nice form (i.e., CNF). The proof is by a reduction from CSAT, which in turn boils down to introducing auxiliary variables in order to cut the computation of a deep circuit into a conjunction of related computations of shallow (i.e., depth-2) circuits (which may be presented as CNF formulae). The aforementioned auxiliary variables hold the possible values of the internal wires of the circuit.

3SAT. Note that the formulae resulting from the latter reduction are in conjunctive normal form (CNF) with each clause referring to three variables (i.e., two corresponding to the input wires of a node/gate and one to its output wire). Thus, the above reduction actually establishes the NP-completeness of 3SAT (i.e., SAT restricted to CNF formula with up to three variables per clause). Alternatively, reduce SAT (for CNF formula) to 3SAT (i.e., satisfiability of 3CNF formula), by replacing long clauses by conjunctions of three-variable clauses using auxiliary variables.

In order to establish the NP-completeness of the search version of the aforementioned problems we need to present a polynomial-time mapping of solutions for the target problem (e.g., SAT) to solutions for the origin problem (e.g., CSAT). Note that such a mapping is typically explicit in the argument establishing the validity of the Karp-reduction.

4.4 NP sets that are neither in P nor NP-complete

Many (to say the least) other NP-sets have been shown to be NP-complete. A very partial list includes Graph 3-Colorability, Subset Sum, (Exact) Set Cover, and the Traveling Salesman Problem. (Hundreds of other natural problems can be found in [3].) Things reach a situation in which people seem to expect any NP-set to be either NP-complete or in $\mathcal{P}$. This naive view is wrong:

Theorem 4 *Assuming $\mathcal{NP} \neq \mathcal{P}$, there exist NP-sets that are neither NP-complete nor in $\mathcal{P}$.*

We mention that some natural problems (e.g., factoring) are conjecture to be neither solvable in polynomial-time nor NP-hard, where a problem Π is NP-hard if any NP-set is reducible to solving Π. See discussion following Theorem 5. We recommend to either state Theorem 4 without a proof or merely provide the proof idea.

Proof idea. The proof is by modifying a set in $\mathcal{NP} \setminus \mathcal{P}$ such that to fail all possible reductions (to this set) and all possible polynomial-time decision procedures (for this set). Specifically, we start with some $L \in \mathcal{NP} \setminus \mathcal{P}$ and derive $L' \subset L$ (which is also in $\mathcal{NP} \setminus \mathcal{P}$) by making each reduction (say of L) to L' fail by dropping finitely many elements from L (until the reduction fails), whereas all possible polynomial-time fail to decide L' (which differ from L only on a finite number of inputs). We use the fact that any reduction (of some set in $\mathcal{NP} \setminus \mathcal{P}$) to a finite set (i.e., a finite subset of L) must fail (and this failure is due to a finite set of queries), whereas any efficient decision procedure for L (or L modified on finitely many inputs) must fail on some finite portion of all possible inputs (of L). The process of modifying L into L' proceeds in iterations, alternatively failing a potential reduction (by dropping sufficiently many strings from the rest of L) and failing a potential decision procedure (by including sufficiently many strings from the rest of L). This can be done efficiently because it is inessential to determine the optimal points of alternation (where sufficiently many strings were dropped (resp., included) to fail a potential reduction (resp., decision procedure)). Thus, L' is the intersection of L and some set in $\mathcal{P}$, which implies that $L' \in \mathcal{NP} \setminus \mathcal{P}$.

5 Three additional topics

The following topics are typically not mentioned in a basic course on complexity. Still, pending on time constraints, we suggest covering them at some minimal level.

5.1 The class coNP and NP-completeness

By prepending the name of a complexity class (of decision problems) with the prefix "co" we mean the class of complement sets; that is,

$$\mathrm{co}\mathcal{C} \stackrel{\mathrm{def}}{=} \{\{0,1\}^* \setminus L : L \in \mathcal{C}\}$$

Specifically, $\mathrm{co}\mathcal{NP} = \{\{0,1\}^* \setminus L : L \in \mathcal{NP}\}$ is the class of sets that are complements of NP-sets. That is, if R is an NP-relation and $L_R = \{x : \exists y\ (x,y) \in R\}$ is the associated NP-set then $\{0,1\}^* \setminus L_R = \{x : \forall y\ (x,y) \notin R\}$ is the corresponding coNP-set.

It is widely believed that $\mathcal{NP}$ is not closed under complementation (i.e., $\mathcal{NP} \neq \text{co}\mathcal{NP}$). Indeed, this conjecture implies $\mathcal{P} \neq \mathcal{NP}$ (because $\mathcal{P}$ is closed under complementation). The conjecture $\mathcal{NP} \neq \text{co}\mathcal{NP}$ means that some coNP-sets (e.g., the complements of NP-complete sets) do not have NP-proof systems; that is, there is no NP-proof system for proving that a given formula is not satisfiable.

If indeed $\mathcal{NP} \neq \text{co}\mathcal{NP}$ then some (non-trivial) NP-sets cannot be Karp-reducible to coNP-sets.[7] However, all NP-sets are reducible to coNP-sets (by a straightforward Cook-reduction that just flips the answer), and so the non-existence of Karp-reduction does not seem to represent anything really fundamental. In contrast, we mention that $\mathcal{NP} \neq \text{co}\mathcal{NP}$ implies that some NP-sets cannot be reduced to sets in the intersection $\mathcal{NP} \cap \text{co}\mathcal{NP}$ (even under general (i.e., Cook) reductions). Specifically,

Theorem 5 *If $\mathcal{NP} \cap \text{co}\mathcal{NP}$ contains an NP-hard set then $\mathcal{NP} = \text{co}\mathcal{NP}$.*

Recall that a set is NP-hard if every NP-set is reducible to it (possibly via a general reduction). Since $\mathcal{NP} \cap \text{co}\mathcal{NP}$ is conjectured to be a proper superset of $\mathcal{P}$, it follows (using the conjecture $\mathcal{NP} \neq \text{co}\mathcal{NP}$) that there are NP-sets that are neither in $\mathcal{P}$ nor NP-hard (i.e., specifically, the sets in $(\mathcal{NP} \cap \text{co}\mathcal{NP}) \setminus \mathcal{P}$). Notable candidates are sets related to the integer factorization problem (e.g., the set of pairs (N, s) such that s has a square root modulo N that is a quadratic residue modulo N and the least significant bit of s equals 1).

Proof: Suppose that $L \in \mathcal{NP} \cap \text{co}\mathcal{NP}$ is NP-hard. Given any $L' \in \text{co}\mathcal{NP}$, we will show that $L' \in \mathcal{NP}$. We will merely use the fact that L' reduces to L (which is in $\mathcal{NP} \cap \text{co}\mathcal{NP}$). Such a reduction exists because L' is reducible $\overline{L}' \stackrel{\text{def}}{=} \{0,1\}^* \setminus L'$ (via a general reduction), whereas $\overline{L}' \in \mathcal{NP}$ and thus is reducible to L (which is NP-hard).

To show that $L' \in \mathcal{NP}$, we will present an NP-relation, R', that characterizes L' (i.e., $L' = \{x : \exists y\ (x, y) \in R'\}$). The relation R' consists of pairs of the form $(x, ((z_1, \sigma_1, w_1), ..., (z_t, \sigma_t, w_t)))$, where on input x the reduction of L' to L accepts after making the queries $z_1, ..., z_t$, obtaining the corresponding answers $\sigma_1, ..., \sigma_t$, and for every i it holds that if $\sigma_i = 1$ then w_i is an NP-witness for $z_i \in L$, whereas if $\sigma_i = 0$ then w_i is an NP-witness for $z_i \in \{0,1\}^* \setminus L$.

We stress that we use the fact that both L and $\overline{L} \stackrel{\text{def}}{=} \{0,1\}^* \setminus L$ are NP-sets, and refer to the corresponding NP-witnesses. Note that R' is indeed an NP-relation: The length of solutions is bounded by the running-time of the reduction (and the corresponding NP-witnesses). Membership in R' is decided by checking that the sequence of (z_i, σ_i)'s matches a possible query-answer sequence in an accepting execution of the reduction[8] (ignoring the correctness of the answers), and that all answers (i.e., σ_i's) are correct. The latter condition is easily verified by use of the corresponding NP-witnesses. ■

5.2 Optimal search algorithms for NP-relations

The title of this section sounds very promising, but our guess is that the students will be less excited once they see the proof. We claim the existence of an *optimal search algorithm for any NP-relation.*

[7] Recall that the empty set cannot be Karp-reducible to $\{0,1\}^*$. Thus, the current assertion refers to (non-trivial) NP-sets. Now, suppose that L Karp-reduces to $L' \in \text{co}\mathcal{NP}$, which means that $\overline{L} \stackrel{\text{def}}{=} \{0,1\}^* \setminus L$ Karp-reduces to $L'' \stackrel{\text{def}}{=} \{0,1\}^* \setminus L' \in \mathcal{NP}$. Then $\overline{L} \in \mathcal{NP}$ by virtue of the NP-relation $\{(x, y) : (f(x), y) \in R''\}$, where R'' is the witness relation of L''. It follows that $L \in \text{co}\mathcal{NP}$.

[8] That is, we need to verify that on input x, after obtaining the answers $\sigma_1, ..., \sigma_{i-1}$ to the first $i-1$ queries, the i^{th} query made by the reduction equals z_i.

Furthermore, we will explicitly present such an algorithm, and prove that it is optimal in a very strong sense: for any algorithm correctly solving the same search problem, it holds that up-to some fixed additive polynomial term (which may be disregarded in case the NP-problem is not solvable in polynomial-time), our algorithm is at most a constant factor slower than the other algorithm. That is:

Theorem 6 *For every NP-relation R there exists an algorithm A that satisfies the following:*

1. *A correctly solves the search problem of R.*
2. *There exists a polynomial p such that for every algorithm A' that correctly solves the search problem of R and for every $x \in L_R$ it holds that $t_A(x) = O(t_{A'}(x) + p(|x|))$, where t_A* (resp., *$t_{A'}$*) *denotes the number of steps taken by A* (resp., *A'*) *on input x.*

We stress that the hidden constant in the O-notation depends only on A', but in the following proof the dependence is exponential in the length of the description of algorithm A' (and it is not known whether a better dependence can be achieved). On the other hand, the optimality of algorithm A refers only to inputs that have a solution (i.e., $x \in L_R$). Interestingly, we establish the optimality of A without knowing what its (optimal) running-time is. Thus, the P-sv-NP Question is "reduced" to determining the running time of a single explicitly presented algorithm (i.e., the optimal algorithm A).

Proof sketch: Fixing R, we let M be a polynomial-time algorithm that decides membership in R, and let p be a polynomial bounding the running-time of M. We present the following algorithm A that merely runs all possible search algorithms "in parallel" and checks the results provided by each of them (using M), halting whenever it obtains a correct solution.

Since there are infinitely many possible algorithms, we should clarify what we mean by "running them all in parallel". What we mean is to run them at different rates such that the infinite sum of rates converges to 1 (or any other constant). Specifically, we will run the i^{th} possible algorithm at rate $1/(i+1)^2$. Note that a straightforward implementation of this idea may create a significant overhead, involved in switching frequently from the computation of one machine to another. Instead we present an alternative implementation that proceeds in iterations. In the j^{th} iteration, for $i = 1, ..., 2^{j/2}$, we emulate $2^j/(i+1)^2$ steps of the i^{th} machine. Each of these emulations is conducted in one chunk, and thus the overhead of switching between the various emulations is insignificant (in comparison to the total number of steps being emulated). We stress that in case some of these emulations halts with output y, algorithm A invokes M on input (x, y) and output y if and only if $M(x, y) = 1$. Furthermore, the verification of a solution provided by a candidate algorithm is also emulated at the expense of its step-count. (Put in other words, we augment each algorithm with a canonical procedure (i.e., M) that checks the validity of the solution offered by the algorithm.)

(In case we want to guarantee that A also halts on $x \notin L_R$, we may let it run an exhaustive search for a solution, in parallel to all searches, and halt with output $\perp$ in case this exhaustive search fails.)

Clearly, whenever $A(x)$ outputs y (i.e., $y \neq \perp$) it must hold that $(x, y) \in R$. To show the optimality of A, we consider an arbitrary algorithm A' that solves the candid search problem of R. Our aim is to show that A is not much slower than A'. Intuitively, this is the case because the overhead of A results from emulating other algorithms (in addition to A'), but the total number of emulation steps wasted (due to these algorithms) is inversely proportional to the rate of algorithm A', which in turn is exponentially related to the length of the description of A'. The punch-line is that since A' is fixed, the length of its description is a constant. ■

5.3 Promise Problems

Promise problems are a natural generalization of decision problems (and search problems can be generalized in a similar manner). In fact, in many cases, promise problems provide the more natural formulation of a decision problem. Formally, promise problems refer to a three-way partition of the set of all strings into yes-instances, no-instances and instances that violate the promise. Standard decision problems are obtained as a special case by insisting that all inputs are allowed (i.e., the promise is trivial), but intuitive formulations of decision problems reads like "given a planar graph, determine whether or not ..." (i.e., the promise is that the input represents a planar graph).

We comment that the aforementioned discrepancy can be easily addressed in the case that there exists an efficient algorithm for determining membership in the "promise set" (i.e., the set of instances that satisfy the promise). In this case, the promise problem is computationally equivalent to deciding membership in the set of yes-instances. However, in case the promise set is not tractable, the terminology of promise problems is unavoidable. Examples include the notion of "unique solutions" and the formulation of "gap problems" as capturing various approximation tasks. For a recent survey on promise problems and their applications, the reader is referred to [5].

Historical Notes

Many sources provide historical accounts of the developments that led to the formulation of the *P vs NP Problem* and the development of the theory of NP-completeness (see, e.g., [3]). We thus refrain from attempting to provide such an account.

One technical point that we mention is that the three "founding papers" of the theory of NP-completeness (i.e., [1, 6, 8]) use the three different terms of reductions used above. Specifically, Cook uses the general notion of polynomial-time reduction [1], often referred to as Cook-reductions. The notion of Karp-reductions originates from Karp's paper [6], whereas its augmentation to search problems originates from Levin's paper [8]. It is worth noting that unlike Cook and Karp's works, which treat decision problems, Levin's work is stated in terms of search problems.

The existence of NP-sets that are neither in P nor NP-complete (i.e., Theorem 4) was proven by Ladner [7], Theorem 5 was proven by Selman [9], and the existence of optimal search algorithms for NP-relations (i.e., Theorem 6) was proven by Levin [8]. (Interestingly, the latter result was proved in the same paper in which Levin presented the discovery of NP-completeness, independently of Cook and Karp.) Promise problems were explicitly introduced by Even, Selman and Yacobi [2].

References

[1] S.A. Cook. The Complexity of Theorem Proving Procedures. In *3rd STOC*, pages 151–158, 1971.

[2] S. Even, A.L. Selman, and Y. Yacobi. The Complexity of Promise Problems with Applications to Public-Key Cryptography. *Inform. and Control*, Vol. 61, pages 159–173, 1984.

[3] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

[4] O. Goldreich. *Introduction to Complexity Theory – notes for a one-semester course.* Weizmann Institute of Science, Spring 2002. Available from `http://www.wisdom.weizmann.ac.il/~oded/cc.html`

[5] O. Goldreich. On Promise Problems: A Survey. This volume.

[6] R.M. Karp. Reducibility among Combinatorial Problems. In *Complexity of Computer Computations*, R.E. Miller and J.W. Thatcher (eds.), Plenum Press, pages 85–103, 1972.

[7] R.E. Ladner. On the Structure of Polynomial Time Reducibility. *Jour. of the ACM*, 22, 1975, pages 155–171.

[8] L.A. Levin. Universal Search Problems. *Problemy Peredaci Informacii 9*, pages 115–116, 1973. Translated in *problems of Information Transmission 9*, pages 265–266.

[9] A. Selman. On the structure of NP. *Notices Amer. Math. Soc.*, Vol 21 (6), page 310, 1974.