Introduction to Complexity Theory*
Lecture 1: The P vs NP Question

Notes taken by Eilon Reshef

November 1998

Summary: We review the fundamental question of computer science, known as $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$: given a problem whose solution can be verified efficiently (i.e., in polynomial time), is there necessarily an efficient method to actually *find* such a solution? First, we define the notion of $\mathcal{NP}$, i.e., the class of all problems whose solution can be verified in polynomial-time. Next, we discuss how to represent search problems in the above framework.

We conclude with the notion of *self-reducibility*, relating the hardness of determining whether a feasible solution exists to the hardness of actually finding one.

1 Introduction

Whereas the research in complexity theory is still in its infancy, and many more questions are open than closed, many of the concepts and results in the field have an extreme conceptual importance and represent significant intellectual achievements.

Of the more fundamental questions in this area is the relation between different flavors of a problem: the *search* problem, i.e., finding a feasible solution, the *decision* problem, i.e., determining whether a feasible solution exists, and the *verification* problem, i.e., deciding whether a given solution is correct.

To initiate a formal discussion, we assume basic knowledge of elementary notions of computability, such as Turning machines, reductions, polynomial-time computability, and so on.

2 The Complexity Class $\mathcal{NP}$

$\boxed{\text{3-5}} \Rightarrow$

In this section we recall the definition of the complexity class $\mathcal{NP}$ and overview some of its basic properties. Recall that the complexity class $\mathcal{P}$ is the collection of all languages L that can be recognized "efficiently", i.e., by a deterministic polynomial-time Turing machine. Whereas the traditional definition of $\mathcal{NP}$ associates the class $\mathcal{NP}$ with the collection of languages that can be efficiently recognized by a *non-deterministic* Turning machine, we provide an alternative definition, that in our view better captures the conceptual contents of the class.

Informally, we view $\mathcal{NP}$ as the class of all languages that admit a short "certificate" for membership in the language. Given this certificate, called a *witness*, membership in the language can be verified efficiently, i.e., in polynomial time.

* Lecture Notes for a course given by Oded Goldreich at the Weizmann Institute of Science, Israel.

For the sake of self-containment, we recall that a (binary) relation R is *polynomial-time decidable* if there exists a polynomial-time Turing machine that accepts the language $\{E(x,y) \mid (x,y) \in R\}$, where $E(x,y)$ is a unique encoding of the pair (x,y). An example of such an encoding is $E(\sigma_1 \cdots \sigma_n, \tau_1 \cdots \tau_m) \triangleq \sigma_1\sigma_1 \cdots \sigma_n\sigma_n 01 \tau_1\tau_1 \cdots \tau_n\tau_n$.

We are now ready to introduce a definition of $\mathcal{NP}$.

Definition 1 *The complexity class $\mathcal{NP}$ is the class of all languages L for which there exists a relation $R_L \subseteq \{0,1\}^* \times \{0,1\}^*$, such that*

- *R_L is polynomial-time decidable.*
- *There exists a polynomial b_L such that $x \in L$ if and only if there exists a* witness *w, $|w| \leq b_L(|x|)$ for which $(x,w) \in R_L$.*

Note that the polynomial bound in the second condition is required despite the fact that R_L is polynomial-time decidable, since the polynomiality of R_L is measured with respect to the length of the pair (x,y), and not with respect to $|x|$ only.

It is important to note that if x is not in L, there is no polynomial-size witness w for which $(x,w) \in R_L$. Also, the fact that $(x,y) \notin R_L$ does *not* imply that $x \notin L$, but rather that y is not a proper witness for x.

A slightly different definition may sometimes be convenient. This definition allows only *polynomially-bounded* relations, i.e.,

Definition 2 *A relation R is* polynomially bounded *if there exists a polynomial $p(\cdot)$, such that for every $(x,y) \in R$, $|y| \leq p(|x|)$.*

Since a composition of two polynomials is also a polynomial, any polynomial in $p(|x|)$, where p is a polynomial, is also polynomial in $|x|$. Thus, if a polynomially-bounded relation R can be decided in polynomial-time, it can also be decided in time polynomial in the size of first element in the pair $(x,y) \in R$.

Now, definition 1 of $\mathcal{NP}$ can be also formulated as:

Definition 3 *The complexity class $\mathcal{NP}$ is the class of all languages L for which there exists a* polynomially-bounded *relation $R_L \subseteq \{0,1\}^* \times \{0,1\}^*$, such that*

- *R_L is polynomial-time decidable.*
- *$x \in L$ if and only if there exists a* witness *w, for which $(x,w) \in R_L$.*

In this view, the fundamental question of computer science, i.e., $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$ can be formulated as the question whether the *existence* of a short witness (as implied by membership in $\mathcal{NP}$) necessarily brings about an efficient algorithm for finding such a witness (as required for membership in $\mathcal{P}$).

To relate our definitions to the traditional definition of $\mathcal{NP}$ in terms of a non-deterministic Turning machine, we show that the definitions above indeed represent the same complexity class.

Proposition 2.1 *$\mathcal{NP}$ (as in definition 1) $=$ $\mathcal{NP}$ (as in the traditional definition).*

$\boxed{6\text{-}7} \Rightarrow$ **Proof:**

First, we show that if a language L is in $\mathcal{NP}$ according to the traditional definition, then it is also in $\mathcal{NP}$ according to definition 1.

Consider a non-deterministic Turing machine $\tilde{M}_L$ that decides on L after at most $p_L(|x|)$ steps, where p_L is some polynomial depending on L, and x is the input to $\tilde{M}_L$. The idea is that one can encode the non-deterministic choices of $\tilde{M}_L$, and to use this encoding as a witness for membership in L. Namely, $\tilde{M}_L$ can always be assumed to first make all its non-deterministic choices (e.g., by writing them on a separate tape), and then execute deterministically, branching according to the choices that had been made in the first step. Thus, $\tilde{M}_L$ is equivalent to a deterministic Turning machine M_L accepting as input the pair (x, y) and executing exactly as $\tilde{M}_L$ on x with a pre-determined sequence of non-deterministic choices encoded by y. An input x is accepted by $\tilde{M}_L$ if and only if there exists a y for which (x, y) is accepted by M_L.

The relation R_L is defined to be the set of all pairs (x, y) accepted by M_L.

Thus, $x \in L$ if and only if there exists a y such that $(x, y) \in R_L$, namely if there exists an accepting computation of M_L. It remains to see that R_L is indeed polynomial-time decidable and polynomially bounded. For the first part, observe that R_L can be decided in polynomial time simply by simulating the Turing machine M_L on (x, y). For the second part, observe that $\tilde{M}_L$ is guaranteed to terminate in polynomial time, i.e., after at most $p_L(|x|)$ steps, and therefore the number of non-deterministic choices is also bounded by a polynomial, i.e., $|y| \leq p_L(|x|)$. Hence, the relation R_L is polynomially bounded.

For the converse, examine the witness relation R_L as in definition 1. Consider the polynomial-time deterministic Turing machine M_L that decides on R_L, i.e., accepts the pair (x, y) if and only if $(x, y) \in R_L$. Construct a non-deterministic Turning machine $\tilde{M}_L$ that given an input x, guesses, non-deterministically, a witness y of size $b_L(|x|)$, and then executes M_L on (x, y). If $x \in L$, there exists a polynomial-size witness y for which $(x, y) \in R_L$, and thus there exists a polynomial-time computation of $\tilde{M}_L$ that accepts x. If $x \notin L$, then for every polynomial-size witness y, $(x, y) \notin R_L$ and therefore $\tilde{M}_L$ always rejects x. ■

3 Search Problems

$\boxed{9} \Rightarrow$

Whereas the definition of computational power in terms of languages may be mathematically convenient, the main computational goal of computer science is to solve "problems". We abstract a computation problem Π by a *search problem* over some binary relation R_Π: the input of the problem at hand is some x and the task is to find a y such that $(x, y) \in R_\Pi$ (we ignore the case where no such y exists).

A particularly interesting subclass of these relations is the collection of *polynomially verifiable* relations R for which

- R is polynomially bounded. Otherwise, the mere writing of the solution cannot be carried out efficiently.
- R is polynomial-time recognizable. This captures the intuitive notion that once a solution to the problem is given, one should be able to verify its correctness efficiently (i.e., in polynomial time). The lack of such an ability implies that even if a solution is provided "by magic", one cannot efficiently determine its validness.

Given a polynomially-verifiable relation R, one can define the corresponding language $L(R)$ as the set of all words x for which there *exists* a solution y, such that $(x, y) \in R$, i.e.,

$$L(R) \triangleq \{x \mid \exists y \; (x, y) \in R\}. \tag{1}$$

By the above definition, $\mathcal{NP}$ is exactly the collection of the languages $L(R)$ that correspond to search problems over polynomially verifiable relations, i.e.,

$$\mathcal{NP} \triangleq \{L(R) \mid R \text{ is polynomially verifiable}\}$$

Thus, the question $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$ can be rephrased as the question whether for every polynomially verifiable relation R, its corresponding language $L(R)$ can be decided in polynomial time.

$\boxed{10} \Rightarrow$ Following is an example of a computational problem and its formulation as a search problem.

PROBLEM: 3-Coloring Graphs
INPUT: An undirected graph $G = (V, E)$.
TASK: Find a 3-coloring of G, namely a mapping $\varphi : V \to \{1, 2, 3\}$ such that no adjacent vertices have the same color, i.e., for every $(u, v) \in E$, $\varphi(u) \neq \varphi(v)$.

The natural relation R_{3COL} that corresponds to 3-Coloring is defined over the set of pairs (G, φ), such that $(G, \varphi) \in R_{3COL}$ if

- φ is indeed a mapping $\varphi : V \to \{1, 2, 3\}$.
- For every $(u, v) \in E$, $\varphi(u) \neq \varphi(v)$.

Clearly, with any reasonable representation of φ, its size is polynomial in the size of G. Further, it is easy to determine in polynomial time whether a pair (G, φ) is indeed in R_{3COL}.

The corresponding language $L(R_{3COL})$ is the set of all 3-colorable graphs, i.e., all graphs G that have a legal 3-coloring.

Jumping ahead, it is $\mathcal{NP}$-hard to determine whether such a coloring exists, and hence, unless $\mathcal{P} = \mathcal{NP}$, no efficient algorithm for this problem exists.

4 Self Reducibility

$\boxed{16} \Rightarrow$ Search problems as defined above are "harder" than the corresponding decision problem in the sense that if the former can be carried out efficiently, so can the latter. Given a polynomial-time search algorithm $\mathcal{A}$ for a polynomially-verifiable relation R, one can construct a polynomial-time decision algorithm for $L(R)$ by simulating $\mathcal{A}$ for polynomially many steps, and answering "yes" if and only if $\mathcal{A}$ has terminated and produced a proper y for which $(x, y) \in R$.

Since much of the research in complexity theory evolves around decision problems, a fundamental question that naturally arises is whether an efficient procedure for solving the *decision* problem guarantees an efficient procedure for solving the *search* problem. As will be seen below, this is not known to be true in general, but can be shown to be true for any $\mathcal{NP}$-complete problem.

We begin with a definition that captures this notion:

Definition 4 *A relation R is* self-reducible *if solving the search problem for R is Cook-reducible to deciding the corresponding language $L(R) \triangleq \{x \mid \exists y\ (x, y) \in R\}$.*

Recall that a Cook reduction from a problem Π_1 to Π_2 allows a Turing machine for Π_1 to use Π_2 as an oracle (polynomially many times).

Thus, if a relation R is self-reducible, then there exists a polynomial-time Turing machine that solves the search problem (i.e., for each input x finds a y such that $(x, y) \in R$), except that the Turning machine is allowed to access an oracle that decides $L(R)$, i.e., for each input x' outputs whether there exists a y' such that $(x', y') \in R$. For example, in the case of 3-colorability, the search algorithm is required to *find* a 3-coloring for an input graph G, given as an oracle a procedure that tells *whether* a given graph G' is 3-colorable. The search algorithm is not limited to ask the oracle only about G, but rather may query the oracle on a (polynomially long) sequence of graphs G', where the sequence itself may depend upon answers to previous invocations of the oracle.

We consider the example of SAT.

$\boxed{17}$ ⇒ PROBLEM: SAT
INPUT: A CNF formula φ over $\{x_1, \ldots, x_n\}$.
TASK: Find a satisfying assignment σ, i.e., a mapping $\sigma : \{1, \ldots, n\} \to \{T, F\}$, such that $\varphi(\sigma(1), \ldots, \sigma(n))$ is true.

The relation R_{SAT} corresponding to SAT is the set of all pairs (φ, σ) such that σ is a satisfying assignment for φ. It can be easily verified that the length of σ is indeed polynomial in n and that the relation can be recognized in polynomial time.

Proposition 4.1 *R_{SAT} is self-reducible.*

$\boxed{18}$ ⇒

Proof:

We show that R_{SAT} is self-reducible by showing an algorithm that solves the search problem over R_{SAT} using an oracle $\mathcal{A}$ for deciding $SAT \triangleq L(R_{SAT})$. The algorithm incrementally constructs a solution by building partial assignments. At each step, the invariant guarantees that the partial assignment can be completed into a full satisfying assignment, and hence when the algorithm terminates, the assignment satisfies φ. The algorithm proceeds as follows.

- Query whether $\varphi \in$ SAT. If the answer is "no", the input formula φ has no satisfying assignment.

- For i ranging from 1 to n, let $\varphi_i(x_{i+1}, \ldots, x_n) \triangleq \varphi(\sigma_1, \ldots, \sigma_{i-1}, 1, x_{i+1}, \ldots, x_n)$. Using the oracle, test whether $\varphi_i \in SAT$. If the answer is "yes", assign $\sigma_i \leftarrow 1$. Otherwise, assign $\sigma_i \leftarrow 0$. Clearly, the partial assignment $\sigma(1) = \sigma_1, \ldots, \sigma(i) = \sigma_i$ can still be completed into a satisfying assignment, and hence the algorithm terminates with a true assignment.

■

Consequently, one may deduce that if SAT is decidable in polynomial time, then there exists an efficient algorithm that solves the search problem for R_{SAT}. On the other hand, if SAT is not decidable in polynomial time (which is the more likely case), there is no efficient algorithm for solving the search problem. Therefore, research on the complexity of deciding SAT relates directly to the complexity of searching R_{SAT}.

In the next lecture we show that every $\mathcal{NP}$-complete language has a self-reducible relation. However, let us first discuss the problem of graph isomorphism, which can be easily shown to be in $\mathcal{NP}$, but is not known to be $\mathcal{NP}$-hard. We show that nevertheless, graph isomorphism has a self-reducible relation.

$\boxed{19}$ ⇒ PROBLEM: Graph Isomorphism
INPUT: Two simple[1] graphs $G_1 = (V, E_1)$, $G_2 = (V, E_2)$. We may assume, without loss of generality, that none of the input graphs has any isolated vertices
TASK: Find an isomorphism between the graphs, i.e. a permutation $\varphi : V \to V$, such that $(u, v) \in E_1$ if and only if $(\varphi(u), \varphi(v)) \in E_2$.

The relation R_{GI} corresponding to the graph isomorphism problem is the set of all pairs $((G_1, G_2), \varphi)$ for which φ is an isomorphism between G_1 and G_2.

[1]Such graphs have no self-loops and no parallel edges, and so each vertex has degree at most $|V| - 1$.

Proposition 4.2 *R_{GI} is self-reducible.*

20-21 **Proof:**

⇒ To see that graph isomoprphism is self-reducible, consider an algorithm that uses a graph-isomorphism membership oracle along the lines of the algorithm for SAT. Again, the algorithm fixes the mapping $\varphi(\cdot)$ vertex by vertex.

At each step, the algorithm fixes a single vertex u in G_1, and finds a vertex v such that the mapping $\varphi(u) = v$ can be completed into a graph isomorphism. To find such a vertex v, the algorithm tries all candidate mappings $\varphi(u) = v$ for all unmapped $v \in V$, using the oracle to tell whether the mapping can still be completed into a complete isomorphism. If there exists an isomorphism to begin with, such a mapping must exist, and hence the algorithm terminates with a complete isomorphism.

We now show how a partial assignment can be decided by the oracle. The trick here is that in order to check if u can be mapped to v, one can "mark" both vertices by a unique pattern, say by rooting a star of $|V|$ leaves at both u and v, resulting in new graphs G_1', G_2'. Next, query the oracle whether there is an isomorphism φ between G_1' and G_2'. Since the degrees of u and v are strictly larger than the degrees of other vertices in G_1' and G_2', an isomorphism φ' between G_1' and G_2' would exist if and only if there exists an isomorphism φ between G_1 and G_2 that maps u to v.

After the mapping of u is determined, proceed by incrementally marking vertices in V with stars of $2|V|$ leaves, $3|V|$ leaves, and so on, until the complete mapping is determined. ■

A point worth mentioning is that the definition of self-reducibility applies to relations and not to languages. A particular language $L \in \mathcal{NP}$ may be associated with more than one search problem, and the self-reducibility of a given relation R (or the lack thereof) does not immediately imply self-reducibility (or the lack thereof) for a different relation R' associated with the same language L.

22 ⇒ It is believed that not every language in $\mathcal{NP}$ admits
a self-reducible relation. Below we present an example of a language in $\mathcal{NP}$ for which the "natural" search problem is believed not to be self-reducible. Consider the language of composite numbers, i.e.,

$$L_{COMP} \triangleq \{N \mid N = n_1 \cdot n_2 \quad n_1, n_2 > 1\}.$$

The language L_{COMP} is known to be decidable in polynomial time by a randomized algorithm. A natural relation R_{COMP} corresponding to L_{COMP} is the set of all pairs $(N, (n_1, n_2))$ such that $N = n_1 \cdot n_2$, where $n_1, n_2 > 1$. Clearly, the length of (n_1, n_2) is polynomial in the length of N, and since R_{COMP} can easily be decided in polynomial time, L_{COMP} is in $\mathcal{NP}$.

However, the search problem over R_{COMP} requires finding a pair (n_1, n_2) for which $N = n_1 \cdot n_2$. This problem is computationally equivalent to factoring, which is believed not to admit any (probabilistic) polynomial-time algorithm. Thus, it is very unlikely that R_{COMP} is (random) self-reducible.

Another language whose natural relation is believed not to be self-reducible is L_{QR}, the set of all quadratic residues. The language L_{QR} contains all pairs (N, x) in which x is a quadratic residue modulo N, namely, there exists a y for which $y^2 \equiv x \pmod{N}$. The natural search problem associated with L_{QR} is R_{QR}, the set of all pairs $((N, x), y)$ such that $y^2 \equiv x \pmod{N}$. It is well-known that the search problem over R_{QR} is equivalent to factoring under randomized reductions. Thus, under the assumption that factoring is "harder" than deciding L_{QR}, the natural relation R_{QR} is not (random) self-reducible.

Bibliographic Notes

43 ⇒ For a historical account of the "P vs NP Question" see [2].

The discussion regarding Quadratic Residiousity is taken from [1]. This paper contains also further evidence to the existence of NP-relations which are not self-reducible.

1. M. Bellare and S. Goldwasser, "The Complexity of Decision versus Search", *SIAM Journal on Computing*, Vol. 23, pages 97–119, 1994.

2. Sipser, M., "The history and status of the P versus NP problem", *Proc. 24th ACM Symp. on Theory of Computing*, pages 603–618, 1992.