## Lecture Series I

# Things that should have been taught in previous courses

The first three lectures focus on material that should have been taught in the basic course on computability. Unfortunately, in many cases this material is covered but from a wrong perspective or without any (proper) perspective. Thus, although in a technical sense most of the material (e.g., the class $\mathcal{NP}$ and the notion of NP-completeness) may be known to the students, its conceptual meaning may not have been appreciated (and our aim is to try to correct this damage).

In addition, we cover some topics that may be new to most students. 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, and the existence of optimal search algorithms for NP-relations.

# Lecture 1

# P versus NP

We assume that all students have heard of P and NP, 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.** Discuss the association of efficiency with polynomial-time. (Polynomials are merely a "closed" set of moderately growing functions, where "closure" means closure under addition, multiplication and composition.)

## 1.1 The search version

We focus on polynomially-bounded relations. The 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 polynomially-bounded condition guarantees that intrinsic intractability may not be due to the length (or mere typing) of the required solution.

**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 (i.e., there exists a polynomial-time algorithm that given $x$ find $y$ such that $(x, y) \in R$ or state that no such $y$ exists).

**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 verify whether 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 recognize a valid solution seems to be a natural prerequisite for a discussion regarding finding such solutions. (Indeed, formally speaking, one can introduce non-NP-relations for which the search problem is solvable in polynomial-time; but 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 every NP-relation can be solved in polynomial-time?* In other words, if it is easy to test whether a solution for an instance is correct then is it also easy to find solutions to given instances? 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. 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. In such a case, the world is more interesting: some reasonable problems are easy to solve whereas others are hard to solve.

## 1.2 The decision version

For an NP-relation $R$, we denote the set of instances having solution by $L_R$; that is, $L_R = \{x : \exists y\ (x, y) \in R\}$. Such a set is called an NP-set. 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 given adequate proofs (i.e., a corresponding NP-witness $y$ such that $(x, y) \in R$).

**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.) An NP-relation $R$ yields a natural verification procedure, which amounts to checking whether 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 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. Recall that $\mathcal{P}$ is the class of sets that can be decided efficiently (i.e., by a polynomial-time algorithm). Then 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 determine membership by ourselves).

## 1.3 Conclusions

Verify that $\mathcal{P} \neq \mathcal{NP}$ *in terms of search problems* if and only if $\mathcal{P} \neq \mathcal{NP}$ *in terms of decision problems*. Thus, it suffices to focus on the latter (simpler) formulation.

Note that $\mathcal{NP}$ is typically defined as the class of sets that can be decided by a fictitious device called a non-deterministic polynomial-time machine. The reason that this class of fictitious devices is important is because it captures (indirectly) the definition of NP-proofs. Verify that indeed the "standard" 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).

# Lecture 2

# 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) reduction among computational problems, and view the notion of a Karp-reduction as an important special case that suffices (and is more convenient) in some cases.

## 2.1 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 standard 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.

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.

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), 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). Likewise, each decision problem in $\mathcal{P}$ is (trivially) reducible to any computational problem (i.e., by a reduction that does not use the subroutine at all), whereas such a trivial reduction is disallowed by the syntax of Karp-reductions.

We comment that Karp-reductions may and should be augmented also 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 $R'$ on input $x'$) obtaining $y'$ such that $(x', y') \in R'$, and uses $y'$ to compute a solution $y$ to $x$ (i.e.,

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

## 2.2 Self-reducibility of search problems

The search problem for $R$ 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. Let $R_{SAT}$ be the set of pairs $(\phi, \tau)$ such that $\tau$ is a satisfying assignment to the formulae $\phi$. Note that $R_{SAT}$ is an NP-relation (i.e., it is polynomially-bounded and easy to decide (by evaluating a Boolean expression)).

**Proposition 2.1** ($R_{SAT}$ is self-reducible): *The search problem $R_{SAT}$ is reducible to $SAT$.*

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

A key point is that the formulae $\phi'$ can be simplified to contain no constants such that they fit the canonical definition of SAT. 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). ■

A similar reduction can be presented also for other NP-complete problems. Consider, for example, 3-Colorability. Note that, in this case, the process of getting rid of constants (representing partial solutions) is more involved. Details are left as an exercise. In general, if you don't see a "natural" self-reducibility process for some NP-complete relation, you should still know that a self-reduction process (alas maybe not a natural one) does exist.

**Theorem 2.2** *Every NP-relation of an NP-complete set is self-reducible.*

**Proof:** Let $R$ be an NP-relation of the NP-complete set $L_R$. Then, we combine the following sequence of 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 2.1).
3. The decision problem $SAT$ is reducible to the decision problem $L_R$ (by the NP-completeness of the latter).

The theorem follows. ■

# Lecture 3

# NP-completeness

This is the third (and last) lecture devoted to material that the students have heard. Again, most students did see an exposition of the technical material in some undergraduate class, but they might have missed important conceptual points. Specifically, we stress that the mere existence of NP-complete sets (regardless if this is SAT or some other set) is amazing.

## 3.1 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 use arbitrary reductions), except that the restricted notion suffices for all positive results and is easier to work with.

We say that a polynomially-bounded relation is NP-complete if it is an NP-relation and every NP-relation is reducible to it.

The mere fact that we have defined something (i.e., NP-completeness) does not mean that this thing exists. It is indeed remarkable that NP-complete problems do exist!

## 3.2 The existence of NP-complete problems

**Theorem 3.1** *There exist NP-complete relations and sets.*

**Proof:** The proof (as well as all NP-completeness) is based on the observation that some NP-relations are "rich enough" to encode all NP-relations. This is most obvious for the "universal" 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 $|y| \leq t$, we 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 indeed $L_U \stackrel{\text{def}}{=} \{z : \exists y\ (z, y) \in R_U\}$ is an NP-set.

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 $L_R = \{x : \exists y\ (z, y) \in R\}$ 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 $(x, y) \in$), let $M_R$ be a polynomial-time machine deciding membership (of alleged $(x, y)$ pairs) in $R$, and let $t_R$ 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 $R_U$ 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 $(\langle M_R, x, 1^{t_R(|x|+p_R(|y|))}\rangle, y) \in R_U$. ■

## 3.3 CSAT and SAT

Define Boolean circuits (directed acyclic graphs with vertices labeled by Boolean operation). Prove the NP-completeness of the circuit satisfaction problem (CSAT). The proof boils down to encoding possible computations of a Turing machine by a corresponding layered circuit, where each layer represents a configuration of the machine, and the conditions of consecutive configurations are captured by uniform local gadgets in the circuit.

Define Boolean formulae (i.e., a circuit with 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 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 CNFs).

## 3.4 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. 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 3.2** *Assuming $\mathcal{NP} \neq \mathcal{P}$, there exist NP-sets that are neither NP-complete nor in $\mathcal{P}$.*

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 (while making only a finite number of possible 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 reduction (by dropping sufficiently many strings from the rest of $L$) and failing a decision procedure (by including sufficiently many strings from the rest of $L$). This can be done efficiently because it is inessential to determine the first location where we have enough strings as long as we determine some location where we have enough.

We mention that some natural problems (e.g., factoring) are conjecture to be neither solvable in polynomial-time nor NP-hard. See discussion following Theorem 3.3.

## 3.5 NP, 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., $\mathcal{NP} \neq \mathrm{co}\mathcal{NP}$). Indeed, this conjecture implies $\mathcal{P} \neq \mathcal{NP}$ (because $\mathcal{P}$ is closed under complementation), and is implied by the conjecture that $\mathcal{NP} \cap \mathrm{co}\mathcal{NP}$ is a proper superset of $\mathcal{P}$. The conjecture $\mathcal{NP} \neq \mathrm{co}\mathcal{NP}$ 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 $\mathcal{P} \neq \mathcal{NP}$ then some (non-trivial) NP-sets cannot be Karp-reducible to coNP-sets (exercise: why). (Recall that the empty set cannot be Karp-reducible to $\{0,1\}^*$.) In contrast, all NP-sets are reducible to coNP-sets (by a straightforward general reduction that just flips the answer). A less obvious fact is that $\mathcal{NP} \neq \mathrm{co}\mathcal{NP}$ implies that some NP-sets cannot be reduced to sets in $\mathcal{NP} \cap \mathrm{co}\mathcal{NP}$ (even under general reductions). Specifically,

**Theorem 3.3** *If $\mathcal{NP} \cap \mathrm{co}\mathcal{NP}$ contains an NP-hard set then $\mathcal{NP} = \mathrm{co}\mathcal{NP}$.*

Recall that a set is NP-hard if every NP-set is reducible to it (possibly via a general reduction). Since $\mathcal{NP} \cap \mathrm{co}\mathcal{NP}$ is conjectured to be a proper superset of $\mathcal{P}$, it follows (using the conjecture $\mathcal{NP} \neq \mathrm{co}\mathcal{NP}$) that there are NP-sets are neither in $\mathcal{P}$ nor NP-hard. Notable examples 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 $L \in \mathcal{NP} \cap \mathrm{co}\mathcal{NP}$ is NP-hard. Given any $L' \in \mathrm{co}\mathcal{NP}$, we will show that $L' \in \mathcal{NP}$. We will merely use the fact that $L'$ reduces to $L$ (which is in $\mathcal{NP} \cap \mathrm{co}\mathcal{NP}$). Such a reduction exists because $L'$ is reducible $\overline{L}' \stackrel{\mathrm{def}}{=} \{0,1\}^* \setminus L'$ (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 $\overline{L} \stackrel{\mathrm{def}}{=} \{0,1\}^* \setminus L$ 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 $(z_i, \sigma_i)$'s matches a possible query-answer sequence in an execution of the reduction[1] (regardless of the correctness of the answers), and that all answers (i.e., $\sigma_i$'s) are correct. The latter condition is easily verified by use of the corresponding NP-witness. ■

[1] 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 $i^{\text{th}}$ query made by the reduction equals $z_i$.

## 3.6 Optimal search algorithms for NP-relations

Actually, this section does not relate to NP-completeness, but rather to NP-relations.

The title 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.* 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 3.4** *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 $x$.*

Interestingly, we establish the optimality of $A$ without knowing what its (optimal) running-time is. 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).

**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). Viewed in different terms, for any *unbounded* function $\alpha : \mathsf{N} \rightarrow \mathsf{N}$, we proceed in iterations such that in the $i^{\text{th}}$ iteration we let each of the first $\alpha(i)$ algorithms run for at most $2^i$ steps. In case some of these algorithms halts with output $y$, algorithm $A$ invokes $M$ on input $(x, y)$ and output $y$ if and only if $M(x, y) = 1$. The verification of a solution provided by an 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 stops 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 $(x, y) \in R$. Now suppose $A'$ is an algorithm that solves $R$. Fixing $A'$, for every $x$, let us denote by $t'(x)$ the number of steps taken by $A'$ on input $x$, where $t'(x)$ also accounts for the running time of $M(x)$. Then, the $t(x)$-step execution of $A$ on input $x$ is "covered" by the $i(x)$th iteration of $A$, provided that $i(x) = \max(|A'|, \log_2 t'(x))$, where $|A'|$ denotes the length of the description of $A'$. Thus, the running time of $A$ on input $x$, denoted $t(x)$, is at most $\sum_{j=1}^{i(x)} \alpha(j) \cdot 2^j$, and for sufficintly large $x$ it holds that $t'(x) \geq |A'|$. Using (say) $\alpha(j) = j$, it follows that $t(x) = O(t'(x) \cdot \log t'(x))$, which almost establishes the theorem (while we don't care about establishing it as stated). ■

# 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. 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 use the three different terms of reductions used above. Specifically, Cook uses the general notion of polynomial-time reduction [17], often referred to as Cook-reductions. The notion of Karp-reductions originates from Karp's paper [47], whereas its augmentation to search problems originates from Levin's paper [57]. 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 3.2) was proven by Ladner [55], and the existence of optimal search algorithms for NP-relations (i.e., Theorem 3.4) was proven by Levin [57].