Handout

The Complexity Class NP and NP-Completeness

Proseminar Theoretische Informatik WiSe 2020-21
Institut für Informatik
Freie Universität Berlin

valentinpi

15. Dezember 2020
(neueste Version)

Abstract

This handout presents basic definitions and problems of the complexity class NP. It will then give a short insight into the importance of P=NP and NP-completeness.

NTIME, The Class NP and Verifiers Following the polynomial time class for deterministic computation, this handout now introduces the same class for the concept of nondeterministic machines.

Remark 1. The *nondeterministic* definition of the Turing machine mainly differs in the state transitioning function $\delta\colon Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{ L, N, R \})$.

Definition 1. In analogy to the class TIME, the class *NTIME* of $f\colon \mathbb{N} \to \mathbb{R}^+$ is defined as:

$$\text{NTIME}(f(n)) \coloneqq \{ L \mid \text{There is a nondeterministic TM that decides } L \text{ in } \mathcal{O}(f(n)) \text{ time for an input of size } n \}$$

There are two ways to introduce the class NP, both of which are equivalent (proof see resources):

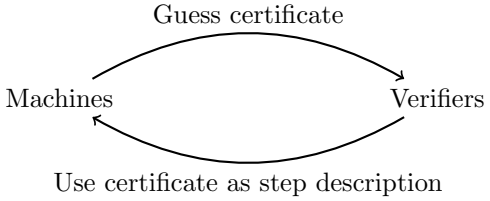
- Through nondeterministic Turing machines (our approach)
- Through polynomial verifiers (an example is included in the following pages)

Definition 2. An algorithm V is called a *verifier* for a language L, if:

$$L = \{ w \mid \exists\, c \in \Sigma^*\colon V \text{ accepts } \langle w, c \rangle \}$$

V is called a *polynomial verifier*, if it runs in polynomial time in terms of $|w|$. L is called *polynomially verifiable*, if such a polynomial verifier exists. c is called *certificate*.

Example: For the PATH problem (see handout P), a verifier could take a directed graph, two nodes and a sequence of edges as an input and check whether the edges match to a path between the nodes.



Definition 3. The class *NP* is the class of languages computable by nondeterministic turing machines in polynomial time.

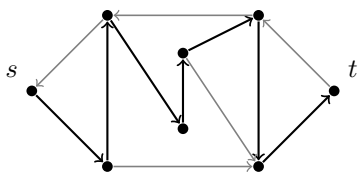
$$\text{NP} \coloneqq \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k)$$

And from the equivalence of both concepts it follows:

Corollary 1. $\text{NP} = \{ L \mid L \text{ has a polynomial time verifier} \}$

NP Problems First, some important terms:

- A *hamiltonian path* between two nodes is a directed path that visits each node exactly once.
- $n \in \mathbb{N}$ is called *composite*, if it is the product of two numbers > 1.
- A *clique* is a subgraph of an undirected graph, where every two nodes are connected. A *k-clique* is a clique of k nodes.



Now consider the following problems:

Theorem 1. HAMPATH $\coloneqq \{ \langle G, s, t\rangle \mid$ There is a hamiltonian path between s and $t \} \in$ NP

Proof. The proof is different from e.g. the PATH problem, since one can now nondeterministically guess a possible path. The following algorithm performs this guess in polynomial time:

Input: $\langle G, s, t\rangle$, where $G = (V, E)$ is a directed graph with nodes s, t.

Function:

1: Index the nodes from 1 to $m \coloneqq |V|$ (number of nodes). Nondeterministically write a list of m numbers $p_1, ..., p_m$, where $p_i \in \{1, ..., m\}$.
2: Check for repetitions. If any are found, reject.
3: Check if p_1, p_m correspond to the node indices of s, t. If not, reject.
4: **for** each $1 \leq i \leq m - 1$ **do**
5: Check if $(p_i, p_{i+1}) \in E$. If not, reject.
6: Accept.

Stage 1 runs in polynomial time due to the nondeterministic choices performed. Stages 2-6 are simple checks. ■

Theorem 2. SUBSET-SUM $\coloneqq \{ \langle S, t\rangle \mid S = \{x_1, ..., x_n\} \wedge \exists \{y_1, ..., y_k\} \subseteq S\colon \sum y_i = t \} \in$ NP
where $\{x_1, ..., x_n\}, \{y_1, ..., y_k\}$ are multisets.

We give an alternative proof to demonstrate the concept of verifiers.

Proof (alternative). We give a polynomial time verifier V. The certificate is the subset.

Input: $\langle \langle S, t\rangle, C\rangle$ with S, C finite set of numbers and c number.

Function:

1: Test if $\sum c_i = t$ with $C = \{c_1, ..., c_k\}$.
2: Test if $C \subseteq S$.
3: Accept if both pass, otherwise reject at one stage.

The checks can be implemented in polynomial time. ■

Two additional, very similar, problems which have been proven to be in NP:

COMPOSITES $= \{ \langle n\rangle \mid n = pq$ with $p, q > 1$ natural numbers $\}$

CLIQUE $= \{ \langle G, k\rangle \mid G$ is an undirected graph with a k-clique $\}$

P=NP Question, Polynomial Time Reduciblity We have observed that:

- P is the class of languages in which membership can be *decided* efficiently.
- NP is the class of languages in which membership can be *verified* efficiently.

Researchers have been unable to prove the existence of a single language in NP that is not in P. Important questions are:

- With nondeterministic behavior, it seems brute-forcing can be avoided. Are there problems which are inherently difficult to solve? Or are researchers not capable to find the polynomial time algorithms for them?
- Even if one could prove that a specific language is in P and NP, how does that influence the P=NP question?

The following concepts of polynomial time reducibility and then NP-completeness give some insight to these questions.

Definition 4. A language A is *polynomial time reducible* to a language B, written $A \leq_p B$, if there is a polynomial time computable function $f\colon \Sigma^* \to \Sigma^*$, where for every $w \in \Sigma^*$:

$$w \in A \leftrightarrow f(w) \in B$$

For our first example of polynomial time reduction consider the problem:

$$\text{3SAT} = \{\ \langle \phi \rangle \mid \phi \text{ is a satisfiable 3cnf-formula } \}$$

Theorem 3. 3SAT $\leq_p$ CLIQUE

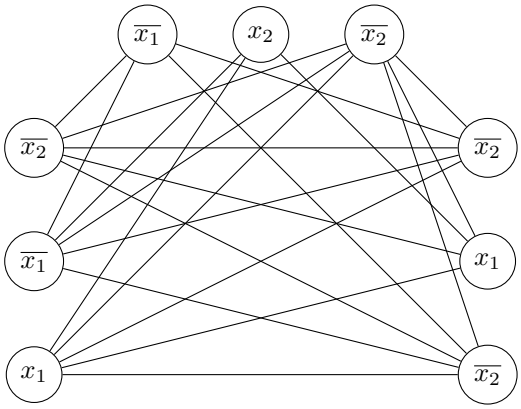
Proof. We present the polynomial reduction by generating an undirected graph G out of the input 3cnf-formula ϕ given.
Let

$$\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_k \vee b_k \vee c_k)$$

be the input 3cnf-formula with exactly k clauses.
Each node is a literal from ϕ's clauses. They are organized in literal groups, corresponding to the clauses, called the *triples* $t_1, \ldots, t_k$. Two nodes are connected if and only if they are not in the same triple and if they are not contradictory, e.g. x and $\overline{x}$.
An example of this construction for the 3cnf-formula

$$\phi = (x_1 \vee \overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_2}) \wedge (\overline{x_2} \vee x_1 \vee \overline{x_2})$$

is illustrated on the right.



We argue that ϕ is satisfiable, if and only if G has a k-clique.

($\Rightarrow$) Suppose ϕ has a satisfying assignment. Then at least one literal is assigned to true in every clause of ϕ. We choose exactly one each. These literals correspond to nodes of G. So by choosing one true literal in every clause of ϕ, a corresponding node is also selected.

- Contrary literals cannot be present in the assignment, since otherwise one clause would be false.
- Each of the literals selected is in a seperate triple. Therefore we have selected exactly k nodes.

Therefore these nodes form a k-clique, because these k literals form an undirected subgraph.

($\Leftarrow$) Suppose G has a k-clique.
The clique must meet the conditions that no contradictory nodes are connected and each node is from a different triple. Therefore, one can assign each variable in the clique a value so that the literal contained is true. So each clause gets assigned *true*, what satisfies the formula.

By that, the reduction is complete. ■

Although these problems are quite different, one can observe similar structures in both that allow the reduction. Through reduction their complexities can be linked.

NP-Completeness It turns out, that the complexities of *all* NP problems are linked by some languages.

Definition 5. A language B is called *NP-complete*, if it satisfies two conditions:

1. $B \in \text{NP}$
2. $A \leq_p B$ for every language $A \in \text{NP}$, also called *NP-hard*

The first language proven to be NP-complete was the SAT problem:

$$\text{SAT} = \{ \langle \phi \rangle \mid \phi \text{ has a satisfying assignment} \}$$

Fact 1 (*Cook-Levin Theorem*)**.** SAT is NP-complete

Proof idea. Showing that SAT $\in$ NP is done by nondeterministically choosing an assignment and checking whether it satisfies the formula.
For the next part it must be shown that any language in NP is polynomial time reducible to SAT. This is done by simulating the turing machine, that computes the language, using a boolean formula. If this formula is satisfiable, the machine can reach an accepting state on the input. This is fundamentally easy to do, but one has to keep track of a lot of details of the simulation, which is why the proof is not discussed here.

By modifying the proof of the Cook-Levin Theorem, one can show that **3SAT is NP-complete**. NP-completeness is relevant, because:

- By proving NP-completeness, one may not waste time searching for a polynomial algorithm that may not exist.
- A lot of practical problems have been proven to be NP-complete.

Corollary 2. If B is NP-complete and $B \in \text{P}$, then $\text{P} = \text{NP}$.

This follows from the definition of NP-completeness: We can use the polynomial time reduction to compute the equivalent problem for B, and then solve it using the polynomial time algorithm.

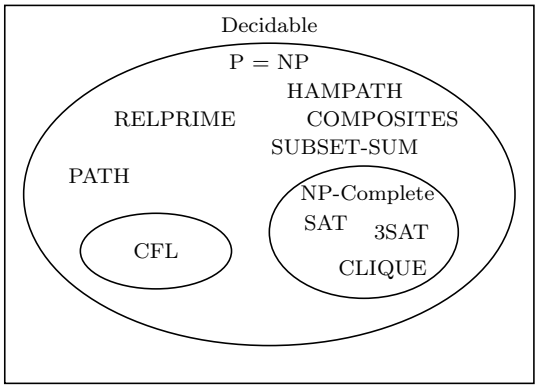
Corollary 3. If B is NP-complete and $B \leq_p C$ for $C \in \text{NP}$, then C is NP-complete.

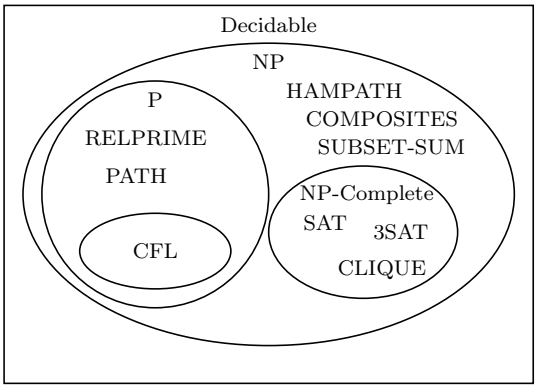
This corollary is the reason, why reduction can be used to easily prove NP-completeness for a lot of languages.

Proof. Since $C \in \text{NP}$, it must be shown that any A is polynomially reducible to C. Since $A \leq_p B$ and $B \leq_p C$, the composition of the polynomial reduction functions lead to $A \leq_p C$. The composition runtime is the sum of two polynomials, so the runtime of the reduction is, again, polynomial. ■

Corollary 4. CLIQUE is NP-complete.

The New Landscape There are two possibilities.





References

[1] Michael Sipser. *Introduction to the Theory of Computation, Third Edition.* Cengage Learning, 2012.

All the illustrations were made by myself using *LaTex/Tikz*.