MTAT.05.125 Theoretical Computer Science

Spring 2022

Vitaly Skachek
Estonian version by Reimo Palm
English version by Yauhen Yakimenka

Lecture 14. NP-complete languages.

Polynomial reductions

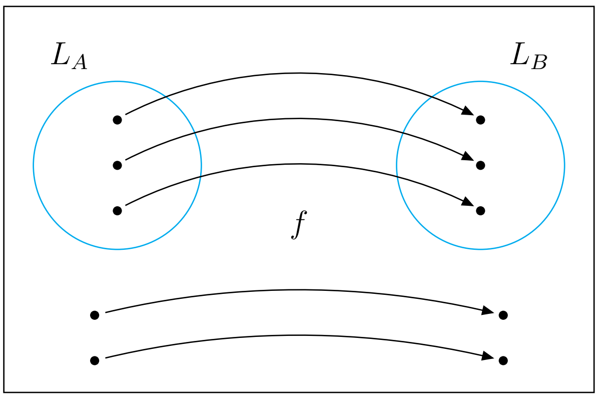
Definition. *A function* $f\colon \Sigma^* \to \Sigma^*$ *is called a* polynomial-time computable *if there exists a Turing machine* M *that halts with just* $f(w)$ *on its tape with running time being polynomial in* $|w|$*, where* w *is input.*

Definition. *A language* L_A *is* polynomial-time mapping reducible *to a language* L_B*, if there exists a polynomial-time mapping reducible function* $f\colon \Sigma^* \to \Sigma^*$*, where for each* w

$$w \in L_A \Leftrightarrow f(w) \in L_B.$$

Denoted: $L_A \leq_P L_B$.

Function f *is called a polynomial-time reduction of* L_A *to* L_B.



Theorem. *If* $L_A \leq_P L_B$ *and* $L_B \in \mathsf{P}$*, then* $L_A \in \mathsf{P}$.

Proof. Let M be the polynomial-time algorithm that decides L_B, and f be the polynomial-time reduction from L_A to L_B. Consider algorithm M', which on input w acts as follows:

1. computes $f(w)$;
2. runs M on input $f(w)$ and outputs according to the output of M.

We have that $w \in L_A$ if and only if $f(w) \in L_B$ (f is a reduction). Equivalently, M accepts $f(w)$ if and only if $w \in L_A$. Equivalently, M' accepts w if and only if $w \in L_A$.

M' runs in polynomial time. Indeed, Step 1 takes polynomial time, and Step 2 also takes polynomial time because composition of two polynomials is a polynomial. □

SAT

A literal is a Boolean variable or a negated Boolean variable, for example: x_1, x_{15}, $\bar{x}_{23}$, z_{10}

A clause is several literals connected with logical OR, for example: $(x_1 \vee \bar{x}_2 \vee x_{15} \vee \bar{x}_{23})$.

A Boolean formula is in *conjunctive normal form*, or a CNF-formula, if it consists of several clauses connected with logical AND, for example:

$$(x_1 \vee \bar{x}_2 \vee x_{15} \vee \bar{x}_{23}) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_7) \wedge (x_3 \vee \bar{x}_{20}).$$

A Boolean formula is a *3-CNF-formula* if it is a CNF-formula and all clauses have three literals, for example:

$$(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_1 \vee x_2 \vee x_{15}) \wedge (x_2 \vee x_3 \vee \bar{x}_{11}).$$

Define languages:

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

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

1. $L \in \mathsf{NP}$*;*
2. *L is* NP*-hard, i.e. any problem $L' \in \mathsf{NP}$ is polynomial-time reducible to L.*

From this definition it follows that if L is NP-complete and $L \in \mathsf{P}$ then $\mathsf{P} = \mathsf{NP}$.

Theorem. *If L is NP-complete and for some $L' \in \mathsf{NP}$ it holds that $L \leq_P L'$, then L' is NP-complete too.*

Proof. It is given that $L' \in \mathsf{NP}$. It remains to show that every $L_0 \in \mathsf{NP}$ is polynomial-time reducible to L'. Since L is NP-complete, $L_0 \leq_{\mathrm{P}} L$. We have:

$$L_0 \leq_{\mathrm{P}} L \qquad \text{ja} \qquad L \leq_{\mathrm{P}} L'.$$

Therefore from definition of polynomial-time reduction we have that $L_0 \leq_{\mathrm{P}} L'$. We obtained that every $L_0 \in \mathsf{NP}$ satisfies $L_0 \leq_{\mathrm{P}} L'$. $\square$

Cooki-Levini teoreem. *SAT is NP-complete.*

Proof idea. First, showing that SAT $\in \mathsf{NP}$ is easy. To show that SAT is NP-hard, we construct polynomial-time reduction from any language $A \in \mathsf{NP}$ to SAT. The reduction takes a string w and produces a Boolean formula ϕ that simulates the non-deterministic machine for A on ϕ. If the machine accepts – there is a satisfying assignment. If the machine rejects – there is no such assignment. Hence $w \in A$ if and only if ϕ is satisfiable.

More details can be found in the book.

Theorem. *3-SAT is NP-complete.*

Proof. First, 3-SAT $\in \mathsf{NP}$, since we can non-deterministically "guess" assignment and verify that it satisfies the given 3-CNF-formula.

Second, let

$$\phi = (l_{1,1} \vee l_{1,2} \vee \ldots \vee l_{1,i_1}) \wedge (l_{2,1} \vee l_{2,2} \vee \ldots \vee l_{2,i_2}) \wedge \ldots \wedge (l_{k,1} \vee l_{k,2} \vee \ldots \vee l_{k,i_k}),$$

where $l_{i,j}$ are literals (can contain "not" sign). We convert each clause into "and" of clauses with 3 literals, as follows:

$$\varphi_j = (l_{j,1} \vee l_{j,2} \vee \ldots \vee l_{j,i_j})$$

is converted into

$$\hat{\varphi}_j = (l_{j,1} \vee l_{j,2} \vee z_1) \wedge (\bar{z}_1 \vee l_{j,3} \vee z_2) \wedge (\bar{z}_2 \vee l_{j,4} \vee z_3) \wedge \ldots \wedge (\bar{z}_{i_j-3} \vee l_{i_j-1} \vee l_{i_j}),$$

where $z_1, z_2, \ldots, z_{i_j-3}$ are new Boolean variables. Then $f(\phi) = \hat{\varphi}_1 \wedge \hat{\varphi}_2 \wedge \ldots \wedge \hat{\varphi}_k$.

Correctness. Prove that $\phi \in$ SAT if and only if $f(\phi) \in$ 3-SAT. In other words, ϕ is satisfiable if and only if $f(\phi)$ is satisfiable.

Direction 1) Let ϕ be satisfiable, and consider assignment to literals that satisfy ϕ. Consider clause $\varphi_j = (l_{j,1} \vee l_{j,2} \vee \ldots \vee l_{j,i_j})$. There exists $l_{j,r}$ in that assignment such that $l_{j,r} = \text{TRUE}$. Then we choose assignments to z_1, …, z_{i_j-3} such that all clauses with three variables are satisfied:

$$(l_{j,1} \vee l_{j,2} \vee \overset{T}{z_1}) \wedge (\overset{F}{\bar{z}_1} \vee l_{j,3} \vee \overset{T}{z_2}) \wedge (\overset{F}{\bar{z}_2} \vee l_{j,4} \vee \overset{T}{z_3}) \wedge \ldots$$

$$\ldots \wedge (\overset{F}{\bar{z}_{r-2}} \vee \overset{T}{l_{j,r}} \vee \overset{F}{z_{r-1}}) \wedge \ldots \wedge (\overset{T}{\bar{z}_{i_j-3}} \vee l_{j,i_j-1} \vee l_{i_j}).$$

Here T and F stand for TRUE and FALSE, respectively.

Direction 2) Let $f(\phi)$ be satisfiable, and consider assignment to $l_{j,r}$ and z_r that satisfy $f(\phi)$. Then, the same assignment to $l_{j,r}$ satisfies ϕ because there are $i_j - 2$ clauses with three variables, but only $i_j - 3$ variables z_r can be TRUE. So at least one of the clauses is satisfied with $l_{j,r} = \text{TRUE}$.

Polynomiality. $f(\phi)$ can be constructed from ϕ in time polynomial in length of ϕ. □

Practise session

1. Define the language

$$\text{SUBSET-SUM} = \{\langle S, t\rangle \mid S = \{x_1, x_2, \ldots, x_k\} \text{ is a set of integer numbers, and for some subset } \{x_{i_1}, x_{i_2}, \ldots, x_{i_l}\} \subseteq S,\ \textstyle\sum_{j=1}^{l} x_{i_j} = t\}.$$

Prove that SUBSET-SUM $\in$ NP.

Solution. Construct non-deterministic Turing machine M, which on input $\langle S, t\rangle$ works as follows:

1. Non-deterministically selects a subset $T \subseteq S$.
2. Checks if $\sum\limits_{x \in T} x = t$.
3. If yes – accepts, if not – rejects.

Correctness. If there exists a subset $\{x_{i_1}, \ldots, x_{i_l}\} \subseteq S$, such that $\sum_{j=1}^{l} x_{i_j} = t$, then the machine M can choose it, and then it accepts (i.e. there exists accepting computation).

If there is no such subset, any choice in Step 1 will lead to rejection.

Polynomiality. Choice in Step 1 requires polynomial time, and also summation in Step 2. Hence, the algorithm has polynomial complexity.

2. Prove that the language

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

is NP-complete.

Solution. First, CLIQUE $\in \mathsf{NP}$ as it is possible to "guess" non-deterministically the clique, and then verify that it has size k.

Next, we show that CLIQUE is NP-hard.

Idea. We use polynomial-time reduction from 3-SAT, which has shown to be NP-complete in the lecture,

$$\text{3-SAT} \leq_{\text{P}} \text{CLIQUE}.$$

Let $\phi = (l_{1,1} \vee l_{1,2} \vee l_{1,3}) \wedge (l_{2,1} \vee l_{2,2} \vee l_{2,3}) \wedge \ldots \wedge (l_{k,1} \vee l_{k,2} \vee l_{k,3})$ be a 3-CNF formula, where $l_{j,r}$ are literals. The reduction f generates a pair $\langle G, k\rangle$, where G is a graph and k is an integer.

The nodes of G are organised in k groups, each group has three nodes. Each such triple correspond to one of the clauses in ϕ, and each node corresponds to a literal in that triple.

More specifically, for clause $(l_{i,1} \vee l_{i,2} \vee l_{i,3})$ we have three nodes in graph, $v_{i,1}$, $v_{i,2}$ and $v_{i,3}$. There is an edge between any two nodes, except when

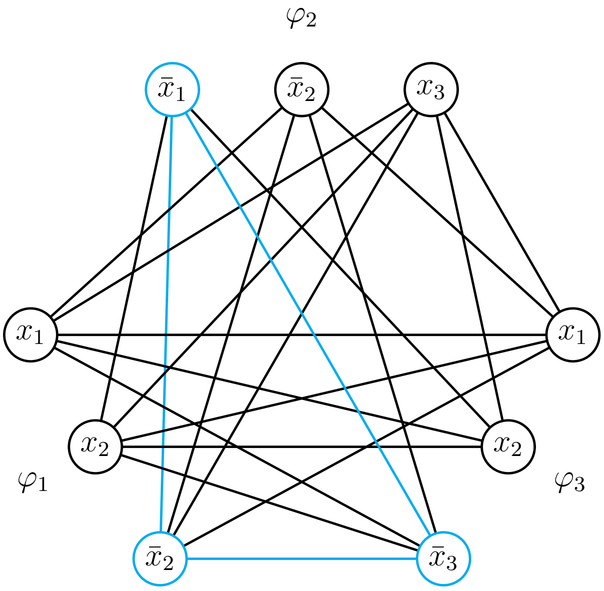
1. two nodes represent literals with contradicting labels, i.e. $l_{j,r}$ and $\bar{l}_{j,r}$, or

2. nodes appear in the same clause.

For example:

$$\phi = \underbrace{(x_1 \vee x_2 \vee \bar{x}_2)}_{\varphi_1} \wedge \underbrace{(\bar{x}_1 \vee \bar{x}_2 \vee x_3)}_{\varphi_2} \wedge \underbrace{(x_1 \vee x_2 \vee \bar{x}_3)}_{\varphi_3}$$

is transformed into a graph



We have a clique of size 3. The corresponding assignment is $\bar{x}_1 =$ TRUE, $\bar{x}_2 =$ TRUE, $\bar{x}_3 =$ TRUE; or $x_1 = x_2 = x_3 =$ FALSE.

Correctness. We need to show that ϕ has a satisfying assignment if and only if G has a clique of size k.

1) Suppose that ϕ has a satisfying assignment. Then, at least one literal is TRUE in every clause. We choose nodes that correspond to those literals. (If more than one literal is TRUE in some clause, we choose one of these literals arbitrarily).

The chosen nodes form a clique because of the following:

- they do not have contradictory values (if x_i is chosen then $\bar{x}_i$ is not chosen);

- they represent different clauses.

The size of this clique is k.

2) Suppose that G has a clique of size k. All nodes of this clique appear in different clauses, because nodes in the same clause are not connected by edges. We assign values to the variables in ϕ such that each literal in the clique has value TRUE. This is always possible since any two literals having contradictory values are not connected by an edge. This assignment satisfies ϕ because each clause contains a clique node, which has value TRUE. Therefore, ϕ is satisfiable.

Polynomiality. Construction of G from ϕ takes only polynomial number of steps.