CS 260
Design and Analysis of Algorithms
7. *P* and *NP*

Mikhail Moshkov

Computer, Electrical and Mathematical Sciences & Engineering Division
King Abdullah University of Science and Technology

P and NP

In this section, we will study the classes P and NP of decision problems, and discuss the notions of NP-hard and NP-complete problems.

If we prove that some problem is NP-hard or NP-complete we will have indirect but serious confirmation of high complexity of the problem.

Classes P and NP

We will work with decision problems for which the output is *yes* or *no*. The input of a decision problem can be encoded as a word in some finite alphabet A.

Let L be a language over the alphabet A which contains only words that corresponds to inputs of the initial decision problem for which the output is *yes*.

Instead of the initial decision problem we will consider the problem of recognition of the language L: for a given word $\alpha \in A^*$ (A^* is the set of all finite words (strings) over A including the empty word λ), it is required to recognize if $\alpha \in L$.

Classes P and NP

Let A and B be two finite alphabets and $\mathcal{A}$ be an algorithm which implements a mapping $\varphi : A^* \to B^*$.

We say that $\mathcal{A}$ is a *polynomial* algorithm if there exists a polynomial $p(n)$ such that, for every natural n, the time of $\mathcal{A}$ work on every input word of the length n is at most $p(n)$.

In particular, it means that for any $\alpha \in A^*$, $|\varphi(\alpha)| \leq |\alpha| + p(|\alpha|)$ where $|\alpha|$ is the length (the number of letters) of the word α.

We can consider only polynomials with nonnegative coefficients, i.e., $p(n)$ is an increasing function.

Classes P and NP

The class P is the class of all languages (decision problems) for each of which there exists a polynomial algorithm for the problem of language recognition.

Classes P and NP

Let A and B be finite alphabets. Let us consider a predicate $Q(x, y) : A^* \times B^* \to \{true, false\}$. We will say that Q is a *polynomial* predicate if there exists a polynomial algorithm that, for given words $\alpha \in A^*$ and $\beta \in B^*$, computes the value $Q(\alpha, \beta)$.

We will say that a language $L \subseteq A^*$ belongs to the class NP if and only if there exists an alphabet B, a polynomial q and a polynomial predicate $Q(x, y) : A^* \times B^* \to \{true, false\}$ such that, for any word $\alpha \in A^*$, we have $\alpha \in L$ if and only if there exists a word $\beta \in B^*$ such that $|\beta| \leq q(|\alpha|)$ and $Q(\alpha, \beta) = true$.

The word β is called the *certificate* for α, and the algorithm for computation of the value $Q(\alpha, \beta)$ is called the *verification* algorithm.

Classes P and NP

Another definition of the class NP is connected with the notion of nondeterministic Turing Machine: L belongs to NP if and only if there exists a nondeterministic Turing Machine which has polynomial time complexity and solves the problem of language L recognition.

Classes P and NP

Theorem 7.1. $P \subseteq NP$.

Proof. Let $L \in P$ and $L \subseteq A^*$. Let $B = \{0\}$ and $q(n) = 1$. We now consider a predicate $Q(\alpha, \beta)$ such that $Q(\alpha, \beta) = \mathit{true}$ if and only if $\alpha \in L$. Since $L \in P$, the predicate Q is a polynomial predicate.

It is clear that $\alpha \in L$ if and only if there exists $\beta \in B^*$ such that $|\beta| \leq 1$ and $Q(\alpha, \beta) = \mathit{true}$ (we can take $\beta = 0$). $\square$

Classes P and NP

We say that a language $L_1 \subseteq A^*$ is *polynomial-time reducible* to language $L_2 \subseteq B^*$ (written $L_1 \leq_P L_2$) if there exists a polynomial time computable function $\varphi : A^* \to B^*$ such that, for all α, $\alpha \in L_1$ if and only if $\varphi(\alpha) \in L_2$.

Classes P and NP

One can prove the following three simple statements:

Proposition 7.2. *If* $L_1 \leq_P L_2$ *and* $L_2 \in P$ *then* $L_1 \in P$*.*

Proposition 7.3. *If* $L_1 \leq_P L_2$ *and* $L_1 \notin P$ *then* $L_2 \notin P$*.*

Proposition 7.4. *If* $L_1 \leq_P L_2$ *and* $L_2 \leq_P L_3$ *then* $L_1 \leq_P L_3$*.*

NP-Hard and NP-Complete Problems

A language L is NP-*hard* if $L' \leq_P L$ for all $L' \in NP$. (Note that it is not necessary to have $L \in NP$.)

The most important properties of NP-hard problems are the following:

- Let L be NP-hard and $L \in P$. Then $NP = P$.
- Let L be NP-hard and $NP \neq P$. Then $L \notin P$.

NP-Hard and NP-Complete Problems

We should note that there are extensions of the notion of NP-hard problem that allow us to use this notion for different kinds of problems, in particular, for optimization problems.

A language L is NP-*complete* if $L \in NP$ and L is NP-hard.

Satisfiability Problem

Conjunctive normal form (CNF) is a Boolean formula of the kind

$$F(x_1, \dots, x_m) = C_1 \wedge C_2 \wedge \dots \wedge C_k$$

where for each $j \in \{1, \dots, k\}$ we have

$$C_j = t_{j1} \vee t_{j2} \vee \dots \vee t_{jn_j}$$

and t_{ji} is either a variable x_p or a negation of variable $\bar{x}_p$.

We assume that each variable occurs in each C_j at most one time. The expression C_j is called *clause*, and t_{ji} is called *literal*.

Satisfiability Problem

Satisfiability problem (SAT): for a given CNF $F(x_1, \ldots, x_n)$ it is required to recognize if there exists a tuple of values $(a_1, \ldots, a_m)$ of variables $x_1, \ldots, x_m$ such that $F(a_1, \ldots, a_m) = 1$.

Let us show that $SAT \in NP$. A certificate for F is a tuple $(a_1, \ldots, a_m)$ of values of variables $x_1, \ldots, x_m$; the verification algorithm checks that $F(a_1, \ldots, a_m) = 1$.

Satisfiability Problem

Theorem 7.5. (Cook, 1971) *SAT is NP-complete problem.*

We now consider other examples of NP-complete problems. To prove that a problem L_1 is NP-complete it is enough to prove that $L_1 \in NP$ and, for some NP-hard problem L_2, we have $L_2 \leq_P L_1$.

Independent Set Problem

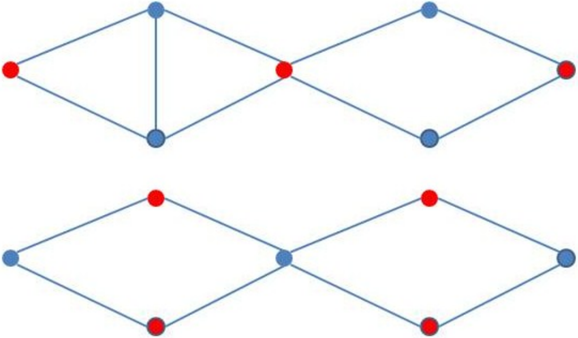
Let $G = (V, E)$ be an undirected graph. We say a set of nodes $S \subseteq V$ is *independent* if no two nodes in S are joined by an edge.

Independent Set problem: for a given undirected graph G and a number k we should recognize if G contains an independent set with at least k nodes.

Independent Set belongs to NP: a certificate is a subset with at least k nodes; the verification algorithm checks, for these nodes, that no edge joins any pair of nodes.

Independent Set Problem

Example 7.6. Independent sets in graphs



Independent Set Problem

To prove that Independent Set is NP-complete we will show that SAT $\leq_P$ Independent Set.

Let $F(x_1, \ldots, x_m)$ be a CNF and $F = C_1 \wedge \ldots \wedge C_k$ where $C_j = t_{j1} \vee t_{j2} \vee \ldots \vee t_{jn_j}$ for $j = 1, \ldots, k$.

We correspond to SAT an Independent Set problem with input $G_F = (V_F, E_F)$, k where G_F is undirected graph with the set of nodes $V_F = \{t_{11}, \ldots, t_{1n_1}, \ldots, t_{k1}, \ldots, t_{kn_k}\}$.

Independent Set Problem

Let $t_{j_1,i_1}, t_{j_2,i_2} \in V_F$. Then the edge $\{t_{j_1,i_1}, t_{j_2,i_2}\}$ belongs to E_F if and only if one of the following conditions holds:

1. $j_1 = j_2$.
2. $j_1 \neq j_2$ and there exists a variable $x_p \in \{x_1, \ldots, x_m\}$ such that $\{t_{j_1,i_1}, t_{j_2,i_2}\} = \{x_p, \bar{x}_p\}$.

Independent Set Problem

It is clear that G_F does not contain independent set with more than k nodes. One can show that G_F contains an independent set with k nodes if and only if there exists a tuple $(a_1, \ldots, a_m)$ of values of variables such that $F(a_1, \ldots, a_m) = 1$.

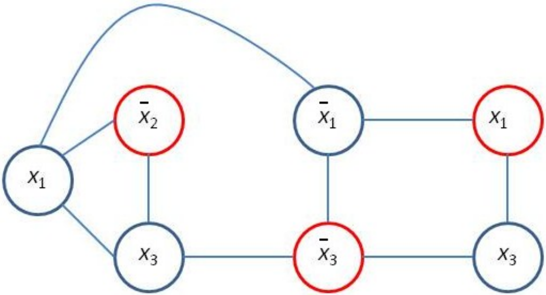
It is not difficult to understand that the considered reduction is a polynomial-time reduction of SAT to Independent Set.

Therefore Independent Set is NP-complete.

Independent Set Problem

Example 7.7. Polynomial-time reduction of SAT to Independent Set

$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (x_1 \vee x_3)$



$x_1 = 1,\ x_2 = 0,\ x_3 = 0$

Vertex Cover Problem

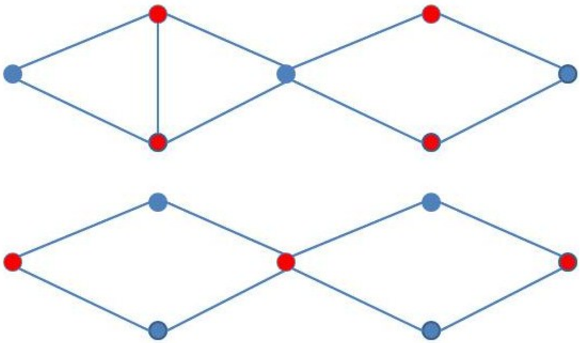
Let $G = (V, E)$ be a undirected graph. We say that a set $S \subseteq V$ is a *vertex cover* if every edge $e \in E$ has at least one end in S.

We formulate the *Vertex Cover problem* as follows: for a graph G and a number k it is required to recognize if G contains a vertex cover with at most k nodes.

Vertex Cover belongs to NP: certificate is a subset with at most k nodes; the verification algorithm checks that each edge has at least one end among these nodes.

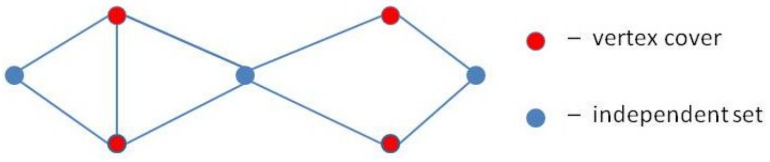
Vertex Cover Problem

Example 7.8. Vertex covers in graphs



Vertex Cover Problem

Let us show that Independent Set $\leq_P$ Vertex Cover. To this end it is enough to prove that a subset $S \subseteq V$ is an independent set if and only if $V \setminus S$ is a vertex cover.



Let S be an independent set. Then, evidently, for each $e \in E$, at least one end of e belongs to $V \setminus S$. Therefore $V \setminus S$ is a vertex cover. Let $V \setminus S$ be a vertex cover. Then there is no edge for which both ends belongs to S, and S is an independent set.

Vertex Cover Problem

From the considered statement it follows that G has an independent set with at least k nodes if and only if G has a vertex cover with at most $|V| - k$ nodes.

Thus, Vertex Cover is NP-complete problem.

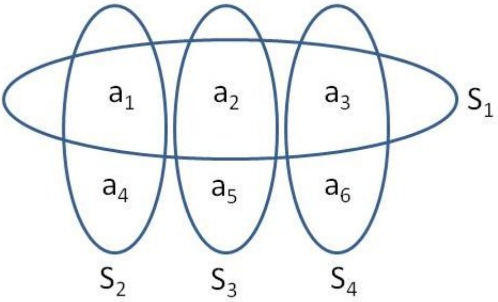
Set Cover Problem

Set Cover problem: for a given set U, a family F of subsets of U such that $U = \bigcup_{S \in F} S$, and a number k, it is required to recognize if there exists a subfamily Q of F with at most k subsets such that $U = \bigcup_{S \in Q} S$.

Set Cover belongs to NP: certificate is a subfamily Q of F with at most k subsets; the verification algorithm checks that every element from U belongs to at least one subset from Q.

Set Cover Problem

Example 7.9. Set cover problem



$U = \{a_1, a_2, a_3, a_4, a_5, a_6\}$, $F = \{S_1, S_2, S_3, S_4\}$

$S_1 = \{a_1, a_2, a_3\}$, $S_2 = \{a_1, a_4\}$, $S_3 = \{a_2, a_5\}$, $S_4 = \{a_3, a_6\}$

Set Cover Problem

Let us show that Vertex Cover $\leq_P$ Set Cover. Let $G = (V, E)$ be an undirected graph.

We correspond to G a Set Cover problem U_G, F_G in the following way: $U_G = E$ and $F_G = \{S_v : v \in V\}$ where S_v is the set of edges from E that have v as an end.

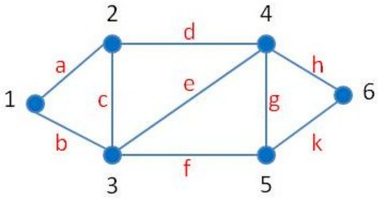
One can show that $\{v_{i_1}, \ldots, v_{i_t}\}$ is a vertex cover for G if and only if

$$\bigcup_{S_v \in \{S_{v_{i_1}}, \ldots, S_{v_{i_t}}\}} S_v = U_G.$$

Thus, Set Cover is NP-complete.

Set Cover Problem

Example 7.10. Polynomial-time reduction of Vertex Cover to Set Cover



$U = \{a, b, c, d, e, f, g, h, k\}$, $F = \{S_1, S_2, S_3, S_4, S_5, S_6\}$,

$S_1 = \{a, b\}$, $S_2 = \{a, c, d\}$, $S_3 = \{b, c, e, f\}$, $S_4 = \{d, e, g, h\}$,
$S_5 = \{f, g, k\}$, $S_6 = \{h, k\}$