

Since SAT and INDEPENDENT SET can be reduced to each other we might think that there would be some similarities between the two problems. In fact, there is one such similarity.

In SAT we want to know if something exists. We are looking for a set of values for to coordinate such that the formula is true. It is hard to find such a set of values but if we have found it, it is easy to check if it makes the formula true.

In INDEPENDENT SET we are looking for a set of nodes of size  $K$  such that the set forms an independent set. It is hard to find the set but if we have found it, it is easy to check if it really is an independent set.

Both the problems have a so called yes-certificate, something that tells us that the answer to the problem is yes. For SAT, the certificate is the values for the variables. For INDEPENDENT SET, the certificate is the  $K$ -set.

Informally, the class NP is the set of decision problems such that if the answer to the problem with input  $x$  is yes, then there is a certificate  $y$ , at most polynomial in the size of  $x$  such that it can be checked in polynomial time (in the size of  $x$ ) that  $y$  is a yes-certificate.

We will give a more formal definition of this. The definition identifies problems with something we will call languages. Then we will describe the property of being an NP-problem as a property for languages.

## Formal definition of P

A *formal language*  $L$  is a set of strings.

Example:

{“abc”, “qwerty”, “xyzy”}

{binary strings of odd length}

{binary strings that represents prime numbers }

{syntactically correct C-programs}

A language can be describe in different ways:

- An enumeration of the strings in the language.
- A set of rules defining the language.
- An algorithm which recognize the strings in the language.

To every decision problem there is a corresponding language:

The language of all yes-instances.

We say that the algorithm  $A$  decides  $L$  if

$$A(x) = \text{Yes if } x \in L,$$

$$A(x) = \text{No if } x \notin L.$$

$A$  runs in *polynomial time* if  $A(x)$  runs in time  $O(|x|^k)$  for all  $x$  and some integer  $k$ .

$$P = \{L : \exists A \text{ that decides } L \text{ i polynomial time}\}$$

## A formal definition of NP

$A$  verifies the instance  $x$  of the problem  $L$  if there is a certificate  $y$  such that  $|y| \in O(|x|^s)$  and

$$A(x, y) = \text{Yes} \iff x \in L$$

This means that  $A$  decides the language

$$L = \{x \in \{0, 1\}^* : \exists y \in \{0, 1\}^* : A(x, y) = \text{Ja}\}$$

$$NP = \{L : \exists A \text{ that verifies } L \text{ in polynomial time}\}$$

$P \overset{NP}{\subseteq}$  since all problem that can be decided in polynomial time also can be verified in polynomial time.

## THE BIG QUESTION

It follows from the definition that  $P \subseteq NP$ .

Is  $P = NP$ ?

Since 1971 this is the most famous open problem in computer science.

Most people believe that the answer is no. Then there must be problems in  $NP - P$ . SAT would be a plausible candidate.

It seems as if hard NP-Problems can be reduced to each other. This observation leads us to the following definition.

NP-Completeness: A problem  $Q$  is NP-Complete if

1.  $Q$  is in NP.
2. For each  $A$  in NP, there is a reduction from  $A$  to  $Q$ , i.e. all NP problems can be reduced to  $Q$ .

Are there any NP-Complete problems? Well, there are:

Cook's Theorem: SAT is NP-Complete

## Other NP-Complete problems

It is easy to see that reductions are transitive, i.e.

$$A \leq B \text{ and } B \leq C \Rightarrow A \leq C$$

We know that  $SAT \leq INDEPENDENT SET$ . We also know that for each  $A$  in  $NP$  we have  $A \leq SAT$ . But this means that for all  $A$  in  $NP$  we have  $A \leq INDEPENDENT SET$

So  $INDEPENDENT SET$  is an  $NP$ -Complete problem.

We realize that the  $NP$ -Complete problems must be the hardest problems in  $NP$ . If any  $NP$ -Complete problem can be solved efficiently then all can!

So we wouldn't expect to be able to find efficient solutions to  $NP$ -Complete problems.

The best way to "show" that a problem is impossible to solve efficiently is to show that it is  $NP$ -Complete.

This is the core of applied Complexity Theory.

But how do we show that a problem is  $NP$ -Complete?

## **A second definition of NP:**

A non-deterministic algorithm is an algorithm that makes random choices. The output is stochastic. We say that  $A$  decides a language  $L$  if:

$$x \in L \Rightarrow A(x) = \text{Yes} \quad \text{with probability } > 0$$

$$x \notin L \Rightarrow A(x) = \text{No} \quad \text{with probability } 1$$

$NP = \{L : \exists \text{ polynomial time non-deterministic algorithm that decides } L\}$

## Proving NP-Completeness

In order to show that  $A$  is NP-Complete it is enough to show that  $A \in NP$  and  $SAT \leq_P A$ .

**Why:** If  $X \in$  we know that  $X \leq SAT$ . If we also have  $SAT \leq A$  we know that  $X \leq A$ ! This shows that  $A$  is NP-Complete.

**Another approach:** We can form a directed graph such that  $A \rightarrow B$  means  $A \leq B$ .

$SAT \rightarrow A \rightarrow B \rightarrow C \rightarrow \dots$  tells us that  $A, B, C, \dots$  are NP-Complete.

To show that  $A$  is NP-Complete we can try to find a known NP-Complete problem  $B$  such that  $B \leq A$ .



## HAMILTONIAN CYCLE $\leq$ TSP

### TSP

Input: A weighted complete graph  $G$  and a number  $K$ .

Goal: Is there a Hamiltonian cycle of length at most  $\leq K$  in  $G$ ?

### HAMILTONIAN CYCLE

Input: A graph  $G$ .

Goal : Is there a Hamiltonian cycle in  $G$ ?

Let  $x = G$  be input to HC. We construct a complete graph  $G'$  with  $w(e) = 0$  if  $e \in G$  and  $w(e) = 1$  if  $e \notin G$ . Then set  $K = 0$ . This will be the input to the TSP.

## Other NP-Complete problems

### Exact Cover

Given a set of subsets of a set  $M$ , is it possible to find a selection of the subsets such that each element in  $M$  is in exactly one of the subsets?

### Subset Sum

Given a set  $P$  of positive integers and an integer  $K$ , is there a subset of the numbers in  $P$  with sum  $K$ ?

### Integer Programming

Given an  $m \times n$ -matrix  $A$ , an  $m$ -vektor  $b$ , an  $n$ -vektor  $c$  and a number  $K$ , is there an  $n$ -vektor  $x$  with integer coefficients such that  $Ax \leq b$  and  $c \cdot x \geq K$ ?

If we relax the condition that the coefficients  $x$  should be integers we get a special case of **Linear Programming**.

## **Subgraph isomorphism is NP-Complete**

**Given two graphs  $G_1$  and  $G_2$ , Is  $G_1$  a subgraph of  $G_2$ ?**

The problem obviously belongs to NP.

We reduce from Hamilton Cycle.

A graph  $G = (V, E)$  contains a Hamiltonian cycle if and only if it contains a subgraph that is a cycle  $C$  with  $|V|$  nodes. So we can set  $G_1 = C$  and  $G_2 = G$ . som  $G$ .

## Partition is NP-Complete

Given a set  $S$  of positive integers.

**Can we split  $S$  into two disjoint parts  $S_1$  and  $S_2$  such that  $\sum_{x \in S_1} x = \sum_{x \in S_2} x$ ?**

The problem is obviously in NP.

We reduce from Subset Sum:

Given an instance  $p_1, p_2, \dots, p_n$  and  $K$  of Subset Sum we create the following instance of Partitioning: Set  $P = \sum p_i$

$$p_1, p_2, \dots, p_n, P - 2K$$

if  $K \leq P/2$  and

$$p_1, p_2, \dots, p_n, 2K - P$$

otherwise.

There is a partitioning precisely when there is a subset sum  $K$ .

## 0/1-programming is NP-Complete

Given an  $m \times n$ -matrix  $A$  and an  $m$ -vektor  $b$ .  
**Is there an  $n$ -vektor  $x$  with coefficients  $\in \{0, 1\}$  such that  $Ax \leq b$ ?**

The problem is in NP since, given  $x$ , we can check in time  $O(n^2)$  if  $Ax \leq b$ .

We reduce from 3-CNF-SAT:

Let  $\Phi$  be an instance of 3-CNF-SAT With  $n$  variables. To each  $x_i$  in  $\Phi$  we define a corresponding variable  $y_i \in \{0, 1\}$  and let 1 Mean True and 0 mean False.

For each clause  $c_j = l_1 \vee l_2 \vee l_3$  we define an inequality

$$T(l_1) + T(l_2) + T(l_3) \geq 1$$

where  $T(x_i) = y_i$  and  $T(\neg x_i) = (1 - y_i)$ .

And that's it!