Intractability

“I can’t find an efficient algorithm, but neither can all these famous people.”
Introduction

After investigating what can be decided, let us focus on what can be computed efficiently, that is, in polynomial time.

The problems that can be solved in polynomial time on a computer coincide with the problems solvable on TMs in polynomial time. This follows from simulation of RAM architecture by a TM, which was not presented in our lectures.

We introduce:

- a new type of reduction
- the theory of intractability

All results are based on the \( \mathcal{P} \neq \mathcal{NP} \), which has not yet been proven or falsified.
1. Classes $P$ and $NP$ : we introduce the theory of intractability and the notion of polynomial time reduction

2. An NP-complete problem : we introduce the SAT problem

3. Restricted version of the satisfiability problem : we introduce a very common variant of SAT

4. Other NP-complete problems : we investigate problems of practical importance that are difficult to solve
A TM $M$ has **time complexity** $T(n)$ if, for any input string $w$ with $|w| = n$, $M$ halts after at most $T(n)$ computational steps.

**Example**: $T(n) = 5n^2 + 3n$, or $T(n) = 4^n + 3n^2$

A language (decision problem) $L$ belongs to the **class $\mathcal{P}$** if there exists a **polynomial** $T(n)$ such that $L = L(M)$ for some (deterministic) TM $M$ with time complexity $T(n)$. 
If the time complexity is not polynomial, we usually say that the time complexity is *exponential*, even if $T(n)$ may not be an exponential function.

**Example**: $T(n) = n^{\log_2 n}$ grows faster than any polynomial, but slower than any exponential $2^{c \cdot n}$.
Polynomial algorithms

The **spanning tree** of a connected graph $G$ is a subset of $G$’s arcs without cycles, that connects all nodes of $G$

The **minimum weight spanning tree** problem (MWST) has as input a graph $G$ with integer weights at its arcs. The problem is to find a spanning tree with the minimum sum of the weights on the arcs
Polynomial algorithms

Example:

The spanning tree with minimum weight is indicated by the arcs in boldface.
Polynomial algorithms

The minimum weight spanning tree can be found using Kruskal algorithm

- for each node \( p \), keep track of the connected component to which \( p \) belongs with respect to the partial spanning tree
- among all arcs that have not yet been processed, consider arc \((p, q)\) with lowest weight; if \((p, q)\) connects two separate connected components
  - add \((p, q)\) to the partial spanning tree
  - merge the two connected components by updating the involved nodes
Polynomial algorithms

Let $m$ be the number of nodes and $e$ be the number of arcs in the graph.

Kruskal algorithm has a very simple implementation on a computer, running in time $O(e(e + m))$

- for each step: choose an arc in time $O(e)$, merge the two components in time $O(m)$
- there are at most $O(e)$ steps

The execution time is therefore **polynomial** in the input size, which we can consider as $(e + m)$
Analyzing the computational complexity of a TM presents two difficulties, as compared to the analysis of a computer algorithm

**Issue 1**: An algorithm can output a structure, while a TM just accepts or rejects its input

We can recast a search problem through a decision problem

**Example**: Given a graph $G$ and an integer $W$, is there a spanning tree with weight not exceeding $W$?

The decision problem usually provides a lower bound for the computational complexity of the search problem, which can be used for intractability to prove that a problem is difficult
Computational complexity analysis

**Issue 2**: Algorithms have input alphabets of unlimited size, while TMs have finite input alphabet.

**Example**: The set of nodes of a graph can be represented by atomic symbols $p_1, \ldots, p_{27}, \ldots, p_{225}$; a TM instead requires some encoding of each symbol, such as $p_1 = p_1$, $p_{27} = p_{11011}$, $\ldots$, $p_{255} = p_{11111111}$.

Symbol encoding introduces a growth factor usually equal to the logarithm of the number of symbols. This factor is not relevant when we study the class of polynomial problems.
A language (decision problem) $L$ belongs to the class $\mathcal{NP}$ if there exists a polynomial function $T(n)$ such that $L = L(M)$ for some NTM $M$ with time complexity $T(n)$.

We can always assume that $M$ performs exactly $T(n)$ moves for every input of length $n$: to this end, we can simulate a clock function on a special tape track.
Nondeterministic polynomial time

\[ \mathcal{P} \subseteq \mathcal{NP} : \text{every TM is also a NTM} \]

\[ \mathcal{P} \neq \mathcal{NP} ? \]

A polynomial NTM can perform an \textit{exponential} number of computations “simultaneously”. Therefore it is commonly assumed that \( \mathcal{P} \neq \mathcal{NP} \), but no one has ever been able to prove this statement.
A **Hamiltonian circuit** in a graph $G$ is a sorting of $G$’s nodes that forms a cycle.

The **traveling salesman problem**, TSP for short, takes as input a graph $G$ with integer weights on the arcs and a weight limit $W$. The problem asks whether $G$ has a Hamiltonian circuit with total weight not exceeding $W$. 
Nondeterministic polynomial time algorithms

In a graph with $m$ nodes, the number of distinct cycles grows with $O(m!)$, which grows faster than $2^{c \cdot m}$

Any deterministic algorithm for TSP seems to need to examine at least an exponential number of cycles and compute the associated weights.

With a nondeterministic algorithm, we can

- choose in each branch of the computation a permutation $\pi$ of the nodes of $G$
- verify the existence of an associated cycle $p_\pi$
- compute the associated weight of $p_\pi$ and compare with $W$

This takes polynomial time, so the problem is in the class $NP$.
To show that a problem $P_2$ cannot be solved in polynomial time, we reduce a problem $P_1 \notin \mathcal{P}$ to $P_2$

We have two issues

- what if the reduction produces an instance of $P_2$ having exponential length in the size of $P_1$?
- what if the reduction takes exponential time?
We impose the additional constraint that the reduction operates in **polynomial time**, and write \( P_1 \leq_p P_2 \)

**Theorem** If \( P_1 \leq_p P_2 \) and \( P_1 \notin \mathcal{P} \) then \( P_2 \notin \mathcal{P} \)

**Proof** If \( P_2 \in \mathcal{P} \), we would have a polynomial time algorithm for \( P_1 \), which is a **contradiction**
A language $L$ is **NP-complete** if

- $L \in \mathcal{NP}$
- for each language $L' \in \mathcal{NP}$ we have $L' \leq_{p} L$

**Example**: TSP is NP-complete (to be proved later)

NP-complete problems are the **most difficult** problems among those in $\mathcal{NP}$

If $\mathcal{P} \neq \mathcal{NP}$ then the NP-complete problems are in $\mathcal{NP} \setminus \mathcal{P}$
NP-complete problems

**Theorem** If $P_1$ is NP-complete, $P_2 \in \mathcal{NP}$, and $P_1 \leq_P P_2$, then $P_2$ is NP-complete.

**Proof** The polynomial time reduction has the transitive property. For any language $L \in \mathcal{NP}$ we have $L \leq_P P_1$ and $P_1 \leq_P P_2$, and therefore $L \leq_P P_2$. 

\[\square\]
Theorem  If an NP-complete problem is in \( \mathcal{P} \), then \( \mathcal{P} = \mathcal{NP} \)

Proof  Assume \( P \) is NP-complete and \( P \in \mathcal{P} \). For any language \( L \in \mathcal{NP} \) we have \( L \leq_p P \) and therefore we can solve \( L \) in polynomial time.

Assuming \( \mathcal{P} \neq \mathcal{NP} \), we consider the proof of NP-completeness of a problem \( P \) as evidence that \( P \notin \mathcal{P} \)
A language $L$ is **NP-hard** if, for each language $L' \in \mathcal{NP}$, we have $L' \leq_p L$

**Note**: We do not require membership in $\mathcal{NP}$. In other words, $L$ could be much more difficult than the problems in $\mathcal{NP}$

**Example**: Some NP-hard problems take exponential time, even if it turns out that $\mathcal{P} = \mathcal{NP}$
We now prove that deciding whether a Boolean expression is satisfiable is an NP-complete problem.

As this is our first NP-complete problem, we must explicitly reduce each problem in \( \mathcal{NP} \) to it.
Boolean expressions are composed by the following symbols:

- an **infinite** set \( \{x, y, z, x_1, x_2, \ldots\} \) of variables defined on Boolean values 1 (true) and 0 (false)
- binary operators \( \wedge \) (logical AND) and \( \vee \) (logical OR)
- unary operator \( \neg \) (logical NOT)
- round brackets (to force precedence)
Boolean expressions

A **Boolean expression** $E$ is recursively defined as

- $E = x$, for any Boolean variable $x$
- $E = E_1 \land E_2$ and $E = E_1 \lor E_2$
- $E = \neg E_1$
- $E = (E_1)$

Operator precedence (decreasing) : $\neg$, $\land$, $\lor$

**Example** : $x \land \neg(y \lor z)$
A **truth assignment** $T$ for a Boolean expression $E$ assigns a Boolean value $T(x)$ (true or false) to each variable $x$ in $E$.

The Boolean value $E(T)$ of $E$ under $T$ is the result of the evaluation of $E$ with each variable $x$ replaced by $T(x)$.

$T$ **satisfies** $E$ if $E(T) = 1$.

$E$ is **satisfiable** if there exists at least one $T$ that satisfies $E$. 
Example

\[ x \land \neg(y \lor z) \text{ is satisfiable: } T(x) = 1, \ T(y) = 0, \ T(z) = 0 \]

\[ x \land (\neg x \lor y) \land \neg y \text{ cannot be satisfied} \]

- we must have \( T(x) = 1 \) and \( T(y) = 0 \)
- therefore \( (\neg x \lor y) \) must be false
The SAT problem

The **satisfiability problem**, SAT for short, is defined as follows

- the input is a Boolean expression $E$
- the output is “yes” if $E$ is satisfiable, “no” otherwise
Boolean expression encoding

We rename the variables as $x_1, x_2, \ldots$ and encode them using symbol $x$ followed by a binary representation of the index.

**Example**: $x_{13} = x1101$

Logical operators and parentheses are represented by themselves.

We have the **alphabet** $\{\land, \lor, \neg, (, ), 0, 1, x\}$ (eight symbols) for encoding of Boolean expressions.

The SAT language is formed by the set of all Boolean expressions that are well-formed, properly coded, and satisfiable.
Example

The Boolean expression

\[ x \land \neg(y \lor z) \]

is encoded as

\[ x1 \land \neg(x10 \lor x11) \]
A Boolean expression $E$ with $m$ occurrences of operators must have $\mathcal{O}(m)$ variable occurrences.

A Boolean expression $E$ of size $m$ has an encoding, written $\text{enc}(E)$, of \textbf{length} $\mathcal{O}(m \log m)$, which is a polynomial function of $m$. 
Theorem SAT is an NP-complete language

Proof (First part) SAT ∈ \( \mathcal{NP} \)

There is a polynomial NTM that solves SAT

- verify that the input is a well formed Boolean expression
- using nondeterminism, guess a truth assignments \( T \); this can be done in polynomial time in the length of \( \text{enc}(E) \)
- for the guessed \( T \), check if \( E(T) = 1 \) and accept accordingly; this can be done in polynomial time in the length of \( \text{enc}(E) \)
Cook Theorem

(Second part) For each $L \in \mathcal{NP}$, $L \leq_p \text{SAT}$

The reduction translates an instance $w$ of the problem represented by $L$ into an instance $E$ of SAT, i.e., a string encoding a Boolean expression.

Let us set a NTM $M$ and a polynomial $p(n)$ such that $L(M) = L$ and $M$ processes $w$ with $|w| = n$ in at most $p(n)$ steps.

In the following $M$ is considered as fixed. The size of $Q$, $\Gamma$ and $\delta$ is therefore considered as a constant.
Cook Theorem

We can assume that

- $M$ has semi-infinite tape and never writes $B$; proof similar to the case of general TM
- on input $w$ with $|w| = n$, $M$ executes exactly $p(n)$ steps on each of its computations; proof uses a clock and extends the $M$ moves by with $\alpha \vdash_{M} \alpha$ for each accepting ID $\alpha$
- all IDs have length $p(n) + 1$ ($p(n)$ symbols and one state); pad the tail of IDs with symbol $B$
Cook Theorem

Let $|w| = n$. Each computation of $M$ on $w$ has the form

$$\gamma = \alpha_0 \leftarrow \alpha_1 \leftarrow \cdots \leftarrow \alpha_p(n)$$

where

- $\alpha_0$ is the initial ID on $w$
- all IDs have the same length
- $\gamma$ accepts if and only if $\alpha_p(n)$ is an accepting ID

Each $\alpha_i$ is represented as a sequence

$$X_{i0}X_{i1} \cdots X_{i,p(n)}$$

where exactly one symbol $X_{ij}$ is a state, and all of the others are tape symbols
### Classes \( \mathcal{P} \) and \( \mathcal{NP} \)

An NP-complete problem

Restricted version of the satisfiability problem

Other NP-complete problems

#### Cook Theorem

\[
\gamma = \alpha_0 \vdash M \alpha_1 \vdash M \cdots \vdash M \alpha_{p(n)}
\]

<table>
<thead>
<tr>
<th>ID</th>
<th>0</th>
<th>1</th>
<th>( \cdots )</th>
<th>( j - 1 )</th>
<th>( j )</th>
<th>( j + 1 )</th>
<th>( \cdots )</th>
<th>( p(n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha_0 )</td>
<td>( X_{00} )</td>
<td>( X_{01} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( X_{0,p(n)} )</td>
</tr>
<tr>
<td>( \alpha_1 )</td>
<td>( X_{10} )</td>
<td>( X_{11} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( X_{1,p(n)} )</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>( \alpha_i )</td>
<td></td>
<td>( X_{i,j-1} )</td>
<td>( X_{i,j} )</td>
<td>( X_{i,j+1} )</td>
<td>( \cdots )</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>( \alpha_{i+1} )</td>
<td></td>
<td>( X_{i+1,j-1} )</td>
<td>( X_{i+1,j} )</td>
<td>( X_{i+1,j+1} )</td>
<td>( \cdots )</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>( \alpha_{p(n)} )</td>
<td>( X_{p(n),0} )</td>
<td>( X_{p(n),1} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( X_{p(n),p(n)} )</td>
</tr>
</tbody>
</table>
Cook Theorem

We represent the computation $\gamma$ using Boolean variables $y_{ij}Z$, where $0 \leq i, j \leq p(n)$, $Z \in (\Gamma \cup Q)$, and

$$y_{ij}Z = \begin{cases} 1, & \text{if } X_{ij} = Z \\ 0, & \text{otherwise} \end{cases}$$

The reduction produces a Boolean expression $E_{M,w}$ such that

- $E_{M,w}$ is satisfiable if and only if there exists an accepting computation of $M$ on $w$
- $E_{M,w}$ can be built in polynomial time in $n$
Cook Theorem

\[ E_{M,w} = U \land S \land N \land F \]

- **U** (uniqueness): only one symbol at each cell
- **N** (next): adjacent ID’s represent a valid move of the TM
- **S** (start): \( \gamma \) starts with the initial ID
- **F** (final): \( \gamma \) halts with an accepting ID
**Cook Theorem**

**Uniqueness:**

\[
U = \bigwedge_{0 \leq i,j \leq p(n)} \neg(y_{ij}z_1 \land y_{ij}z_2)
\]

We have \(O(p(n)^2 \times |\Gamma \cup Q|^2)\) terms. Since we consider \(|\Gamma \cup Q|^2\) as a constant, the number of terms is \(O(p(n)^2)\).

\(|U|\) is a polynomial function of \(n\), where \(n\) is the length of the input instance \(w\).
Cook Theorem

Start: let \( w = a_1 a_2 \cdots a_n \)

\[
S = y_{00} q_0 \land y_{01} a_1 \land y_{02} a_2 \land \cdots \land y_{0n} a_n \land \\
y_{0,n+1} B \land y_{0,n+2} B \land \cdots \land y_{0,p(n)} B
\]

We have \( O(p(n)) \) terms, and \( |S| \) is a polynomial function of \( n \)
Cook Theorem

**Final**: let \( s_1, s_2, \ldots, s_k \) be all the final states of \( M \)

\[
F = \bigvee_{0 \leq j \leq p(n)} \bigvee_{1 \leq h \leq k} y_{p(n)j}s_h
\]

We have \( \mathcal{O}(p(n)) \) terms and \(|F|\) is a polynomial function of \( n \)
Cook Theorem

Next:

\[ N = \bigwedge_{0 \leq i \leq p(n) - 1} N_i \]

Each expression \( N_i \) guarantees that \( \alpha_i \vdash_{M} \alpha_{i+1} \)
In order to check the validity of each relation $\alpha_i \vdash M \alpha_{i+1}$ we always look into windows composed of three tape cells:

<table>
<thead>
<tr>
<th>$X_{i,j-1}$</th>
<th>$X_{i,j}$</th>
<th>$X_{i,j+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_{i+1,j-1}$</td>
<td>$X_{i+1,j}$</td>
<td>$X_{i+1,j+1}$</td>
</tr>
</tbody>
</table>

On the basis of $X_{i,j-1}$, $X_{i,j}$, $X_{i,j+1}$ and of the move chosen by $M$:

- it is **always** possible to check the validity of $X_{i+1,j}$
- in **some cases** ($X_{i,j} \in Q$) it is also possible to check the validity of $X_{i+1,j-1}$ and $X_{i+1,j+1}$
Cook Theorem

\[ N_i = \bigwedge_{0 \leq j \leq p(n)} (A_{ij} \lor B_{ij}) \]

The Boolean expression \( A_{ij} \) states that
- \( X_{ij} \) is the state of \( \alpha_i \)
- \( M \) can choose any move in \( \delta(X_{ij}, X_{i,j+1}) \)

The Boolean expression \( B_{ij} \) states that
- \( X_{ij} \) is not a state
- if the state of \( \alpha_i \) is not \( X_{i,j-1} \) or \( X_{i,j+1} \), then \( X_{i+1,j} = X_{ij} \)

When the state of \( \alpha_i \) is \( X_{i,j-1} \) or \( X_{i,j+1} \), the validity of \( X_{i+1,j} \) is guaranteed by \( A_{i,j-1} \) or \( A_{i,j+1} \)
Cook Theorem

Let $q_1, q_2, \ldots, q_m$ be all of the states of $M$ and let $Z_1, Z_2, \ldots, Z_r$ be all of its tape symbols.

$$B_{ij} = (y_{i,j-1,q_1} \lor y_{i,j-1,q_2} \lor \cdots \lor y_{i,j-1,q_m}) \lor$$
$$\quad (y_{i,j+1,q_1} \lor y_{i,j+1,q_2} \lor \cdots \lor y_{i,j+1,q_m}) \lor$$
$$\quad ((y_{i,j,Z_1} \lor y_{i,j,Z_2} \lor \cdots \lor y_{i,j,Z_r}) \land$$
$$\quad ((y_{i,j,Z_1} \land y_{i+1,j,Z_1}) \lor (y_{i,j,Z_2} \land y_{i+1,j,Z_2}) \lor \cdots \lor$$
$$\quad (y_{i,j,Z_r} \land y_{i+1,j,Z_r}))))

- if the state of $\alpha_i$ is adjacent to $X_{ij}$ we do not impose any condition
- if the state of $\alpha_i$ is $X_{ij}$, $B_{ij}$ is false so that $A_{ij}$ must be true
- if the state of $\alpha_i$ is not $X_{i,j-1}$ or $X_{i,j+1}$, then $X_{i+1,j} = X_{ij}$
Cook Theorem

Let us assume that \((p, C, L) \in \delta(q, A)\) and \(D \in \Gamma\). Then

\[
X_{i,j-1}X_{i,j}X_{i,j+1} = DqA \\
X_{i+1,j-1}X_{i+1,j}X_{i+1,j+1} = pDC
\]

is a valid assignment for the logical variables in a \(2 \times 3\) rectangle in the table representing a computation.

We can represent the assignment by means of the term

\[
Y_{i,j-1,D} \land Y_{i,j,q} \land Y_{i,j+1,A} \land Y_{i+1,j-1,p} \land Y_{i+1,j,D} \land Y_{i+1,j+1,C}
\]
Cook Theorem

Let us assume that \((p, C, R) \in \delta(q, A)\). Then

\[
X_{i,j-1}X_{i,j}X_{i,j+1} = DqA
\]
\[
X_{i+1,j-1}X_{i+1,j}X_{i+1,j+1} = DCp
\]

is a valid assignment

The assignment is represent by means of the term

\[
Y_{i,j-1,D} \land Y_{i,j,q} \land Y_{i,j+1,A} \land Y_{i+1,j-1,D} \land Y_{i+1,j,C} \land Y_{i+1,j+1,p}
\]
An assignment for a $2 \times 3$ rectangle is valid if

1. $X_{i,j} \in Q$ and $X_{i,j-1}, X_{i,j+1} \in \Gamma$
2. there is a move by $M$ that changes the values of $X_{i,j-1}X_{i,j}X_{i,j+1}$ into the values of $X_{i+1,j-1}X_{i+1,j}X_{i+1,j+1}$

The number of valid assignments depends on the size of $Q$ and $\Gamma$ and on the moves in $\delta$. Since $M$ is fixed, the number of valid assignments is a constant

$A_{ij}$ is the logical OR of all terms representing valid assignments
Classes $\mathcal{P}$ and $\mathcal{NP}$
An NP-complete problem
Restricted version of the satisfiability problem
Other NP-complete problems

Cook Theorem

Summarizing:

\[
N = \bigwedge_{0 \leq i \leq p(n)-1} N_i
\]

\[
N_i = \bigwedge_{0 \leq j \leq p(n)} (A_{ij} \vee B_{ij})
\]

$|A_{ij}|, |B_{ij}|$ are costants ($M$ is fixed)

$|N_i|, |N|$ are polynomial functions of $n$

To conclude, $E_{M,w}$ is satisfiable if and only if $w \in L(M)$, and

$|E_{M,w}|$ is a polynomial function of $n$
Normal forms for Boolean expressions

Boolean expressions have a fairly complex structure

We introduce a **simplified** version of SAT, called 3SAT

- 3SAT is an NP-complete problem
- 3SAT is particularly convenient to define reductions that we will investigate later
Normal forms for Boolean expressions

A **literal** is a variable or else the negation of a variable.  
**Example**: \( x; \overline{x} = \neg x \)

A **clause** is the disjunction (logical OR) of literals.  
**Example**: \( x \lor \overline{y} \lor z \)

A Boolean expression in **conjunctive normal form**, or CNF for short, is a conjunction (logical AND) of clauses.  
**Example**: \( (x \lor \overline{y}) \land (\overline{x} \lor z) \)
We use $+$ in place of $\lor$ and we use $\times$ in place of $\land$. As for arithmetic expressions, we represent $\times$ by means of concatenation.

**Example**:

1. $(x \lor \overline{y}) \land (\overline{x} \lor z)$ is written as $(x + \overline{y})(\overline{x} + z)$
2. $(x + y\overline{z})(\overline{x} + y + z)$ is not in CNF
3. $xyz$ is in CNF
Normal forms for Boolean expressions

A Boolean expression is in \textit{k-conjunctive normal form}, or \textit{k-CNF} for short, if

- it is in CNF
- every clause has \textbf{exactly} \textit{k} literals

\textbf{Example} : \((x + \overline{y})(\overline{x} + z)\) is in 2-CNF

We introduce two new decision problems

- CSAT : is some CNF satisfiable ?
- \textit{kSAT} : is some \textit{k}-CNF satisfiable ?
**Theorem** CSAT is NP-complete

**Proof** $\text{CSAT} \in \mathcal{NP}$; $\text{SAT} \leq_p \text{CSAT}$

**Theorem** 3SAT is NP-complete

**Proof** $\text{3SAT} \in \mathcal{NP}$; $\text{CSAT} \leq_p \text{3SAT}$
Finding out that a decision problem is NP-complete indicates that there are very few chances to discover an efficient algorithm for its solution. It is therefore recommended to look for partial / approximate solutions, using heuristics.

The large number of failed attempts to prove $P = NP$ provides evidence that every NP-complete problem requires exponential time for an exact solution.
NP-completeness

Many collections of NP-complete problems have been published and are constantly updated.

Typically, these decision problems are described according to the following scheme:

- problem name and abbreviation
- problem input and its representation / encoding
- specification of positive instances of the problem
- problem used in the reduction for the NP-completeness result
Example

**Problem**: satisfiability of Boolean expressions in 3-CNF (3SAT)

**Input**: Boolean expressions in 3-CNF

**Output**: “yes” if and only if the Boolean expressions is satisfiable

**Reduction**: from CSAT
Independent set

In a graph $G$, a subset $I$ of the nodes is an **independent set** if no pair of nodes in $I$ is connected by some arc of $G$.

An independent set is **maximal** if any other independent set of $G$ has a number of nodes smaller or equal than the former.
Example

$I = \{1, 4\}$ is an independent set

$I$ is maximal: any set of three nodes from the graph has some arc connection
Independent set

The problem of finding a maximal independent set is investigated in the area of *combinatorial optimization*

We consider here the *decision* version of this problem
Independent set

**Problem**: independent set (IS)

**Input**: undirected graph \( G \) and lower bound \( k \)

**Output**: “yes” if and only if \( G \) has an independent set with \( k \) nodes

**Reduction**: from 3SAT

For small values of \( k \), it can be easy to solve the problem. But if \( k \) is the size of the maximal independent set, then the solution of the problem is **difficult**
**Theorem** IS is NP-complete

**Proof** (First part) IS ∈ $\mathcal{NP}$

Let us consider a NTM that

- arbitrarily chooses $k$ nodes using **nondeterminism**
- verifies that the chosen set is independent, and accepts accordingly

The two phases described above can be performed in polynomial time in the size of the input data
IS is NP-complete

(Second part) \(3\text{SAT} \leq_p \text{IS}\)

Let \(E = e_1 \land e_2 \land \cdots \land e_m\) be a Boolean expression in 3-CNF, where each \(e_i\) is a clause.

We build \(G\) with \(3m\) nodes. Each node is identified by a pair \([i, j]\), with \(1 \leq i \leq m\) and \(j \in \{1, 2, 3\}\).

The \([i, j]\) node represents the \(j\)-th literal in the \(i\)-th clause.
Example

\[ E = (x_1 + x_2 + x_3)(\overline{x_1} + x_2 + x_4)(\overline{x_2} + x_3 + x_5)(\overline{x_3} + \overline{x_4} + \overline{x_5}) \]
IS is NP-complete

Construction of graph $G$

- one arc for each pair of nodes in the same column; then one can choose no more than one node per clause
- one arc for each pair of nodes $[i_1,j_1], [i_2,j_2]$, if these represent the literals $x$ and $\overline{x}$; then one cannot choose two literals in an independent set if they are one the negation of the other

We let $k = m$
IS is NP-complete

We can build $G$ and $k$ in polynomial time (quadratic) in the length of the representation of $E$

We prove that $E$ is satisfiable if and only if $G$ has an independent set with $m$ elements
IS is NP-complete

(If part) Let $I$ be an independent set with $m$ elements. We define

- $T(x) = 1$ if the node for $x$ is in $I$
- $T(x) = 0$ if the node for $\overline{x}$ is in $I$
- $T(x)$ can be arbitrarily defined if the nodes for $x$ and $\overline{x}$ are not in $I$

Since nodes for $x$ and $\overline{x}$ cannot simultaneously belong to $I$, the definition of $T$ is consistent.

An independent set $I$ contains exactly one node per clause. It follows that the definition of $T$ satisfies $E$. 

Automata, Languages and Computation

Chapter 10
IS is NP-complete

(Only if part) Let $T$ be an assignment that satisfies $E$. We arbitrarily choose a true literal for each clause, and we add to $I$ the node associated with that literal.

$I$ has size $m$

$I$ is an independent set
- if one arc connects two nodes from the same clause, the two nodes are not both in $I$ by construction
- if the remaining arcs connect two nodes corresponding to a literal and its negation, then the two nodes are not both in $I$ because we have chosen only literals that are true in $T$
Node cover

In a graph $G$, a subset $C$ of the nodes is a **node cover** if each arc of $G$ impinges upon **at least one** node in $C$.

A node cover is **minimal** if its size is smaller or equal than the size of any other node cover of $G$. 
Problem: node cover (NC)
Input: undirected graph $G$ and upper bound $k$
Output: “yes” if and only if $G$ has a node cover with at most $k$ nodes
Reduction: from IS
NC is NP-complete

**Theorem**  NC is NP-complete

**Proof**  (First part)  \( \text{NC} \in \mathcal{NP} \)

Let us consider a NTM that

- arbitrarily chooses \( k \) nodes of the input graph \( G \), using **nondeterminism**
- tests whether the chosen set is a node cover, and accepts accordingly

Both the above steps can be carried out in time polynomial in the size of the input
NC is NP-complete

(Second part) \( IS \leq_p NC \)

Let \( G, k \) be an instance of \( IS \), and let \( n \) be the number of nodes of \( G \). We produce an instance of \( NC \) formed by \( G \) and \( n - k \).

Construction takes \textit{polynomial time}.

We prove that \( G \) has an independent set with \( k \) elements if and only if \( G \) has a node cover with \( n - k \) elements.
(If part) Let $N$ be the set of nodes of $G$ and let $C$ be a node cover with $n - k$ nodes. We argue that $N \setminus C$ with $k$ nodes is an independent set for $G$

Let us assume that $N \setminus C$ is not independent. Then there are nodes $v, w \in (N \setminus C)$ that are connected by some arc.

Thus $v, w \notin C$ and then the arc $(v, w)$ is uncovered. We have a contradiction since $C$ is a node cover.
NC is NP-complete

(Only if part) Let $I$ be an independent set with $k$ nodes. We argue that $N \setminus I$ is a node cover with $n - k$ nodes.

Let us assume that an arc $(v, w)$ is not covered by $N \setminus I$. Then $v, w$ are in $I$.

Since $(v, w)$ is an arc, we have a contradiction because $I$ is an independent set. $\square$
Directed Hamiltonian circuit

Let $G$ be an oriented graph. A **directed Hamiltonian circuit** in $G$ is an oriented cycle that passes through each node of $G$ **exactly once**

**Problem**: directed Hamiltonian circuit (DHC)

**Input**: directed graph $G$

**Output**: “yes” if and only if $G$ has a directed Hamiltonian circuit

**Reduction**: from 3SAT
Undirected Hamiltonian circuit

**Problem**: undirected Hamiltonian circuit (HC)

**Input**: undirected graph $G$

**Output**: “yes” if and only if $G$ has an undirected Hamiltonian circuit

**Reduction**: from DHC
Traveling salesman problem

**Problem** : traveling salesman problem (TSP)

**Input** : undirected graph $G$ with integer weights at every arc, and upper bound $k$

**Output** : “yes” if and only if $G$ has an undirected Hamiltonian circuit such that the sum of the weights at each arc is smaller equal than $k$

**Reduction** : from HC
Summary of our reductions

Classes $\mathcal{P}$ and $\mathcal{NP}$
An NP-complete problem
Restricted version of the satisfiability problem
Other NP-complete problems

All of $\mathcal{NP}$

SAT

CSAT

3SAT

IS DHC

NC HC

TSP

NP

Automata, Languages and Computation
Chapter 10