CS 208: Computing Theory

Prof. Dr. Brahim Hnich
Faculty of Computer Sciences
Izmir University of Economics
Complexity Theory

Time Complexity
A decidable problem is
- Computationally solvable in principle
- But not necessarily in practice

Problem is resource consumption
- Time
- Space
Complexity Theory

Computability theory:
“What is and what is not possible with a computer?”

For the problems that are computable, this leaves the next question: “If we can solve a problem P, how easy or hard is it to do so?”

Complexity theory tries to answer this.
Counting Resources

*Complexity theory is ‘the art of counting resources’.*

**Time complexity:** How many time-steps does the computation of a problem cost?

**Space complexity:** How many bits of memory are required for the computation?

*Again we will use the Turing machine model.*
Measuring Complexity
Just as with computability, the complexity of a problem should take into account all possible Turing machine programs.

Consider the prime decision problem
PRIME = { x | x ∈ N, x is a prime }

The (time) complexity of a specific instance x ∈ PRIME? is meaningless.
(The number can be listed in a big table.)
Asymptotics

The (perceived) difficulty of PRIME is illustrated by the fact that the \( x \in \text{PRIME}? \) question seems to become much more difficult as \( x \) gets bigger.

We will study the relation between the bit-size of a problem instance and the required time/space complexity of the solution for such an instance (worst case).

Asymptotical notions will play a crucial role.
Time Complexity
Let M be a Turing machine that halts on all inputs.

**Definition 7.1**: The *running time* or *time complexity* of M is the function $f: \mathbb{N} \rightarrow \mathbb{N}$, defined by the maximization:

$$f(n) = \max_{|x|=n} (\text{no. of time-steps of } M \text{ on } x)$$

Note: Worst case and size of the input x in bits.

We say: “$f(n)$ is the running time of $M$”, and “$M$ is an $f(n)$ time Turing machine”.
“f(n) = O(g(n))”

It will be extremely convenient to use the following ‘order-notation’ to express our complexities.

**Definition 7.2**: Let f and g be two functions \( N \rightarrow \mathbb{R}^+ \). We say and write \( f(n) = O(g(n)) \) if and only if there are two positive constant c and \( n_0 \) such that \( f(n) \leq c \cdot g(n) \) for all \( n \geq n_0 \).

We say that:

“g(n) is an (asymptotic) upper bound on f(n)”.
Some big O examples

Let \( f(n) = 15n^2 + 7n \) and \( g(n) = \frac{1}{2}n^3 \).

We have \( f(n) = O(g(n)) \) because for \( n_0 = 16 \) and \( c = 2 \), we see indeed for all \( n \geq n_0 \):
\[
\begin{align*}
f(n) &= 15n^2 + 7n \\
&\leq 16n^2 \\
&\leq n^3 = c \cdot g(n).
\end{align*}
\]

More tight is \( 5n^4 + 27n = O(n^4) \).

Take \( n_0 = 1 \) and \( c = 32 \).

(But \( n_0 = 3 \) and \( c = 6 \) works also.)

For polynomials \( p(n) \), it is all a matter of degree…
Polynomials vs Exponentials

Let $f$ be a $k$-th degree polynomial $f(n) = a \cdot n^k + \ldots$, then $f(n) = O(n^r)$ for all $r \geq k$.

Exponential functions like $2^n$ always ‘overpower’ polynomials. For all constants $a$ and $k$, the function $f(n) = a \cdot n^k + \ldots$, will obey: $f(n)=O(2^n)$.

For functions in $n$, we have $n^k = O(b^n)$ for all positive $k$, and $b>1$. 
Logarithms

\[ n = O(n \cdot \log(n)) \] but \[ n \cdot \log(n) = O(n^d) \] for which \( d \)?

**Answer**: For every \( d > 1 \).

Polynomials dominate (powers of)-logarithms, just like exponentials overpower polynomials.

Note that because \( a \cdot \log(n) = O(\log(n)) \) for all \( a \), we do not have to indicate the base \( b \) of \( \log_b \).
The O-Ordering

- $f(n) = O(f(n))$ for all functions $f$.
- $(\log(n))^k = O(n)$ for all constant $k$.
- $n^k = O(2^n)$ for all constant $k$.

We can also put the $O$ in the exponent:

- $f(n) = 2^{O(\log n)}$ thus implies (for big enough $n$):
  - $f(n) \leq 2^{c \cdot \log(n)} = n^c$ for some $c$.
- Equivalent with $f(n) = n^{O(1)}$.  

Little $o$-Notation

**Definition 7.5:** Let $f$ and $g$ be two functions $\mathbb{N} \to \mathbb{R}^+$. We say and write $f(n) = o(g(n))$ if and only

$$\lim_{{n \to \infty}} \frac{f(n)}{g(n)} = 0$$

Where big-O is about “less-than-or-equal”, little-o is about “strictly less than”. 
About Notation

Make it clear whether you mean $O(g(n))$ or $o(g(n))$.

Make it clear in which variable the function is: $O(x^y)$ can be polynomial as well as exponential.

No redundant notation.
Not $O(5n^3 + 6n)$, but simply $O(n^3)$ will do.

Try to keep your $O$ ‘as tight as possible’.
Analyzing Algorithms

Consider a 1-tape TM that decides the language $A = \{ 0^k1^k \mid k=0,1,2,... \}$.

A trivial TM for this $A$ will have quadratic time-complexity: $O(n^2)$ time steps for inputs $y \in \{0,1\}^n$.

A less obvious approach gives a TM with time-complexity $O(n \cdot \log(n))$.

This 2nd TM is asymptotically faster than the 1st.
Definition 7.7: For the function \( t : \mathbb{N} \rightarrow \mathbb{N} \), the time complexity class \( \text{TIME}(t(n)) \) is the following set of decision problems:

\[
\text{TIME}(t(n)) = \{ L \mid \text{there is a 1-tape TM that decides the language } L \text{ in time } O(t(n)) \}
\]

Previous slide: \( A \in \text{TIME}(n \cdot \log(n)) \).
One can also show that: \( A \notin \text{TIME}(n) \).

There is a 2-tape TM that solves \( A \) in time \( O(n) \)…
Model Dependency?

Like with computability, we want the complexity of a problem to be independent of the specific TM-model that we use.

The previous result gives an $O(n \cdot \log(n))$ versus $O(n)$ difference between 1 and 2-tape TMs.

What to do?

Answer: We group $O(n^2)$, $O(n \cdot \log(n))$, $O(n)$, et cetera together in a ‘polynomial-time class’.
The ‘Polynomial Time Class’ P

**Definition 7.11**: The class of languages that can be decided by a single tape TM in polynomial time is denoted by \( P \):

\[
P = \bigcup_{k=1,2,...} \text{TIME}(n^k)
\]

Many, many problems are in \( P \). The problems in \( P \) are **efficiently solvable**.
Theorem 7.8: If a problem \( A \) can be solved in polynomial time on a multi-tape TM, then \( A \) can be solved in \( \text{poly-time} \) on a single-tape TM: \( A \in \mathbb{P} \).

**Proof**: Let \( M \) be the \( k \)-tape TM that solves \( A \) in time \( O(n^d) \).

Theorem 3.8 explained how to simulate the computation of a \( k \)-tape TM by a 1-tape TM \( S \). If \( M \) requires \( t(n) \) time-steps, then this simulation can be done with \( (t(n))^2 \) time steps of \( S \). If \( t(n) = O(n^d) \), then \( S \) solves \( A \) in time \( O(n^{2d}) \). Hence \( A \in \mathbb{P} \).
Robustness of $\mathbf{P}$

In general, every kind of TM (k-tape, r-heads, etc.) can be solved by a standard, single tape TM with only polynomial time/space overhead.

As a result, the poly-time class $\mathbf{P}$ does not depend on the specific computational model that you use.

If some TM can solve $A$ in poly-time, then $A \in \mathbf{P}$.

We can extend the TM-model in many ways…
Strong Church Turing Thesis

All reasonable computational models are polynomial-time/space equivalent:
It is always possible to simulate one model with a machine from another model with only polynomial time/space overhead.

The answer to the question “$A \in \mathbf{P}$?” does not depend on the model that we favor.
Unreasonable Models?

What makes a model ‘unreasonable’?

Partial answer: physical unrealistic requirements for the proper functioning of the machine.

Typical examples:
– **Analog computing:**
  Infinite precision of the elementary components.
– **Unbounded parallel computing:**
  Requires exponential space and energy.
– ...
Nondeterministic TMs

What about nondeterministic machines?

Informal: A nondeterministic TM is able to make ‘lucky guesses’ during its computation.

Computability-wise there is no difference between deterministic and nondeterministic TMs.

For time-complexity, nondeterminism does seem to make difference: NP versus P problem.
Verifiers and NP

Languages in $\textsf{P}$ have poly-time deciders.
Languages in $\textsf{NP}$ have poly-time ‘verifiers’.

**Definition 7.5:** A verifier for a language $A$ is a program $V$ such that we can write $A$ as

$$A = \{<w> \mid V \text{ accepts } <w,c> \text{ for some } c\}$$

(The string $c$ is the certificate that proves $w \in A$.)

**Definition 7.16:** $\textsf{NP}$ is the class of languages that have polynomial time verifiers.
Theorem 7.17: A language L is in NP if and only if L can be decided by a poly-time nondeter. TM.

Proof: Let $A \in \textbf{NP}$ have an $O(n^k)$ time verifier $V$. A polytime NTM can guess the $O(n^k)$ certificate $c$ that $V$ has to use for $x \in A$ and simulate $V$ on $\langle x,c \rangle$.

Let $N$ be the $O(n^k)$ time nondeter. decider for $B$. The $O(n^k)$ guesses of $N$ define a certificate $c$. A polytime deterministic $V$ can simulate $N$ on $\langle x,c \rangle$ and verify “$x \in B$” for every such $x$. 
Corollary 7.19: Let NTIME(t(n)) be the class of languages decidable by a O(t(n)) nondeterministic Turing machine. Then

\[ \text{NP} = \bigcup_{k=1,2,...} \text{NTIME}(n^k) \]

Aside: Most of the times, the certificate/verifier way of looking at NP problems gives more relevant information. Note that only proofs for \( x \in A \) are required, not for the complement \( x \notin A \).
Example CLIQUE

Consider a graph $G$, is there a $k$-clique?

- Graph
- 4-clique

but no 5-clique

$\text{CLIQUE} = \{<G,k> \mid \text{graph } G \text{ has a } k\text{-clique}\}$
Boolean Formulas and SAT

A Boolean variable $x$ can be TRUE or FALSE, which is also denoted by “1” or “0”.

Standard Boolean operations are AND ($x \land y$), OR ($x \lor y$), and NOT ($\neg x$ or also $\overline{x}$).

Typical Boolean formula: $\phi(x,y) = (\neg x \lor y) \land (x \lor \neg y)$. This $\phi$ is satisfiable (by the assignments $x=y=$TRUE and $x=y=$FALSE).

$SAT = \{<\phi> | \phi \text{ is a satisfiable Boolean formula}\}$
Conjunctive Normal Form

Literal = Boolean variable \((x)\) or its negation \((\neg x)\).

Clause = A set of OR-ed literals, like \((x \lor \neg y \lor z)\)

A formula \(\phi\) is in conjunctive normal form (CNF), if it is the AND of clauses. For example:

\[
\phi(x, y, z) = (x \lor \bar{y}) \land (z) \land (\bar{z} \lor y \lor \bar{y})
\]

A 3CNF formula has only clauses with 3 literals:

\[
\phi(x, y, z) = (x \lor \bar{y} \lor \bar{z}) \land (z \lor z \lor z) \land (\bar{z} \lor y \lor \bar{y})
\]

3SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable 3CNF formula}\}
A CNF Example

As a computer scientist you are expected to be fluent in Boolean transformations.

Example: Describe in CNF the Boolean function

\[
\text{ONE}(x_1, \ldots, x_n) = \begin{cases} 
1 & \text{if there is exactly one "true" } x_j \\
0 & \text{otherwise}
\end{cases}
\]

Answer:

\[
\text{ONE}(x_1, \ldots, x_n) = \left( \bigcup_{j=1}^{n} x_j \right) \land \left( \bigcap_{j \neq k} (\overline{x_j} \lor \overline{x_k}) \right)
\]
“CLIQUE = SAT = 3SAT”

We will prove that the languages CLIQUE, SAT and 3SAT are all equally complex when measured with the polynomial time measure stick.

First we will prove that 3SAT is not more difficult than CLIQUE:

“If you know how to solve CLIQUE in polytime, then you also know how to solve 3SAT in polytime”
Mapping Reducibility

Thus far, we used reductions informally: If “knowing how to solve B” implied “knowing how to solve A”, then we had a reduction from A to B.

Here now comes rigor…

(Sounds familiar?)
Poly-Time Functions

A function \( f : \Sigma^* \rightarrow \Sigma^* \) is a poly-time-computable function if there is a TM that on every input \( w \in \Sigma^* \) halts after \( \text{poly}(|w|) \) steps with \( f(w) \) on the tape.

All the usual computations (addition, multiplication, sorting, minimization, etc.) are all poly-time.

Important here is that object transformations, like “Given a formula \( \phi \), make a graph \( G_\phi \)” can almost always be done in poly-time.
A language $A$ is **polynomial time reducible** to another language $B$ if there is a polynomial time computable function $f: \Sigma^* \rightarrow \Sigma^*$ such that:

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

for every $w \in \Sigma^*$.

**Terminology/notation:**
- $A \leq_p B$
- $f$ is the **polynomial time reduction** of $A$ to $B$
- also called: “*polynomial time many-one reducible*”
The language B can be more difficult than A.

Typically, the image $f(A)$ is only a subset of B, and $f(\Sigma^* \setminus A)$ a subset of $\Sigma^* \setminus B$.

“Image $f(A)$ can be the easy part of B”.
Theorem 7.25:
If \( A \leq_{P} B \) and \( B \) is in \( P \), then \( A \) is in \( P \) as well.

Proof: Let \( M \) be the poly-time TM for \( B \) and \( f \) the reducing function from \( A \) to \( B \). Consider the TM:

On input \( w \):
1) Compute \( f(w) \)
2) Run \( M \) on \( f(w) \) and give the same output.

By definition of \( f \): \( w \in A \) if and only if \( f(w) \in B \).
\( M \) “accepts” \( f(w) \) in poly-time if \( w \in A \), and
\( M \) “rejects” \( f(w) \) in poly-time if \( w \notin A \).
Example CLIQUE

Consider a graph $G$, is there a $k$-clique?

CLIQUE = \{<G,k> | graph G has a k-clique\}
Idea 3SAT $\leq_P$ CLIQUE

- Turn every 3CNF formula $\phi(x_1,\ldots,x_n)$ into a graph $G_\phi$

- ... such that the $k$ clauses of $\phi$ can be satisfied with an assignment $\in \{0,1\}^n$...

- ... if and only if $G$ has a $k$-clique.
Example 3SAT to CLIQUE

- Formula (4 clauses, 4 variables):

\[
(x_1 \lor x_2 \lor x_3) \land (\overline{x}_1 \lor x_2 \lor x_4) \land (\overline{x}_2 \lor x_2 \lor \overline{x}_3) \land (x_1 \lor \overline{x}_3 \lor \overline{x}_4)
\]

Draw the graph with 3k nodes, label the literals, completely connect the clauses.

Next, remove the “contradicting edges” like \((x_1, \overline{x}_1)\)
Example 3SAT to CLIQUE

Formula (4 clauses, 4 variables):

\[(x_1 \lor x_2 \lor x_3) \land (\overline{x_1} \lor x_2 \lor x_4) \land (\overline{x_2} \lor \overline{x_2} \lor \overline{x_3}) \land (x_1 \lor \overline{x_3} \lor \overline{x_4})\]

Draw the graph with 3k nodes, label the literals, completely connect the clauses.

Next, remove the “contradicting edges” like \((x_1, \overline{x_1})\)
3SAT to CLIQUE cont.

\[(x_1 \lor x_2 \lor x_3) \land (\overline{x}_1 \lor x_2 \lor x_4) \land (\overline{x}_2 \lor \overline{x}_2 \lor \overline{x}_3) \land (x_1 \lor \overline{x}_3 \lor \overline{x}_4)\]

\[x_1, \overline{x}_2, \overline{x}_3, x_4\]

is a satisfying assignment

\[\overline{x}_1, \overline{x}_2, \overline{x}_3, x_4\]

is not a satisfying assignment

... gives a 4-clique

... gives not a 4-clique
3SAT to CLIQUE Reduction (1)

Let $\phi$ be a 3CNF formula, and $G_{\phi}$ its graph by the previous construction (can be done in poly-time).

If $\phi$ is satisfiable, let $z_1, \ldots, z_n \in \{0, 1\}^n$ be a satisfying assignment.

Pick a satisfying literal in each of the $k$ clauses. In the graph $G_{\phi}$, these $k$ picks will correspond to a $k$-clique because:
- Each triplet contains one pick
- Only the edges between improper assignments ($z_j = 0$ and $z_j = 1$) are removed
3SAT to CLIQUE Reduction (2)

If $G_{\phi}$ has a $k$ clique, then $\phi$ is satisfiable:
The $k$-points of the clique have to be distributed over the $k$ clauses.
Assign the variables $x_j$ according to the points.
- This assignment is proper
- Each clause is satisfied by chosen point.

Hence: $\phi \in 3SAT$ if and only if $G_{\phi} \in k$-CLIQUE
(with $k$ the number of clauses of $\phi$).
Important:
The mapping $\phi \rightarrow G_{\phi}$ is poly-time computable
Definition 7.27: A language B is \textbf{NP}-complete if:
1) B is in NP, and
2) For every language \( A \in \text{NP} \) we have \( A \leq_p B \).

\textbf{NP}-complete problems are the most difficult problems in \textbf{NP}…

If we omit requirement 1), then we sometimes say that B is \textbf{NP}-hard.
Theorem 7.28

If there is an NP-complete problem $B$ that can be decided in deterministic polynomial time, then for all languages $A \in \text{NP}$ we also have $A \in \text{P}$.

**Proof:** If $A \in \text{NP}$, then by the definition of \textbf{NP}-completeness: $A \leq_p B$.

From $B \in \text{P}$, it follows that also $A \in \text{P}$. 
There are \textbf{NP}-complete problems (SAT and 3SAT for example).

We will have to spend some time proving: "2) For every \(A \in \textbf{NP}\) we have \(A \leq_p \text{SAT}\)."

We will use some earlier work on "puzzling".
Proof Outline

Let $A$ be a language in $\mathbf{NP}$.

For every $w$, we want a (CNF) formula $\phi_w$ such that $w \in A \iff \phi_w \in \mathbf{SAT}$, with a poly-time function that calculates $\phi_w$ from $w$.

Let $N$ be the nondeterministic $\mathbf{NP}$ that accepts $A$. Key idea:

$w \in A \iff \exists$ accepting path of $N$ on $w$

\[ \iff \exists x_1 \ldots x_m \ [ \phi_w(x_1 \ldots x_m) = \text{TRUE} ] \]
More Proof Outline
Specifically we will establish the chain:
\( w \in A \iff \exists \text{ accepting path of } N \text{ on } w \)

\( \iff \)

There exists a sequence \( C_1, \ldots, C_T \) of configurations with:
- \( C_1 \) the start configuration of \( N \) on \( w \)
- \( (C_j, C_{j+1}) \) a proper \( N \) transition for every \( j \)
- \( C_T \) an accepting configuration

\( \iff \exists x_1 \ldots x_m \ [ \phi_z(x_1 \ldots x_m) = \text{TRUE} ] \)
Accepting Path of N on w

<table>
<thead>
<tr>
<th>$C_1 =$</th>
<th>#</th>
<th>$q_0$</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>...</th>
<th>$w_n$</th>
<th>_</th>
<th>...</th>
<th>_</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_2 =$</td>
<td>#</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>_</td>
<td>...</td>
<td>_</td>
<td>#</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C_T =$</td>
<td>#</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>$q_A$</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>#</td>
</tr>
</tbody>
</table>
Size of Path of N on w
N is a poly-time nondeterministic TM:
The (accepting) path of N on w is limited by $O(n^k)$.

The sequence of configurations can be described by a tableau $n^k \times n^k$ cells:

$$\phi_w(x_1 \ldots x_m) \quad T=n^k$$

describes this tableau...
How $\phi$ Describes the Tableau
The m Boolean variables in $\phi_w(x_1 \ldots x_m)$ have to describe:
- that each cell is proper
- that $C_1$ is the start configuration of $N$ on $w$
- that the transitions $(C_j, C_{j+1})$ are proper
- that $C_T$ is an accepting configuration
How $\phi$ Describes the Cells

The TM $N$ has state set $Q$ and tape alphabet $\Gamma$. The content of a cell is in the set $Q \cup \Gamma \cup \{\#\}$.

Define the Boolean variables according to:

$$x_{(i,j,s)} = \begin{cases} \text{TRUE} & \text{if } \text{cell}(i, j) = s \\ \text{FALSE} & \text{otherwise} \end{cases}$$

In total there are $n^k \times n^k \times |Q \cup \Gamma \cup \{\#\}|$ Boolean $x$-variables. This is $O(n^{2k})$ polynomial in $n$. 
for Proper Cells

Not all \( x \)-assignments make sense. We require that each cell \((i,j)\) has a one description. For every \( 1 \leq i, j \leq n^k \) there should be only one variable \( x_{(i,j,s)} \) set TRUE. The cell symbols \( s \) range from 1 to \( c = |Q \cup \Gamma \cup \{\#}\| \). Hence we define with the ONE-function:

\[
\phi_{\text{cell}} = \bigcap_{i,j=1}^{n^k} \text{ONE}(x_{(i,j,1)}, \ldots, x_{(i,j,c)})
\]
ϕ_{start} for the Start Configuration

The starting state of N is q_0, and the input string is w_1,…,w_n.
(The rest is filled with _-spaces and marked on the left and right by #-symbols.)

Hence:

$$\phi_{start} = x_{(1,1,\#)} \land x_{(1,2,q_0)} \land$$

$$x_{(1,3,w_1)} \land \cdots \land x_{(1,n+2,w_n)} \land$$

$$x_{(1,n+3,\_)} \land \cdots \land x_{(1,n^k-1,\_)} \land x_{(1,n^k,\#)}$$
After the TM $N$ has entered the accepting state $q_{accept}$, it stays in this state.
We only have to check if the bottom row contains the accepting state:

$$\phi_{accept} = \bigcup_{j=1}^{n^k} x(n^k,j,q_{accept})$$
The last requirement is that the sequence of configurations $C_1, C_2, \ldots, C_T$ are allowed by $N$.

We can check this by locally checking all $2 \times 3$ windows.

There are $(n^k-1) \times (n^k-2)$ of such windows:

$$\phi_{\text{move}} = \bigcap_{i=1}^{n^k-1} \bigcap_{j=1}^{n^k-2} (i, j) \text{ window legal}$$
Legal Windows
If there is no TM head, the tape should remain unchanged:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td></td>
</tr>
</tbody>
</table>

for all $a, b, c \in \Gamma \cup \{\#\}$.

For the transitions $\delta(q_1, a) = \{(q_2, b, R), (q_3, c, L)\}$:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>q₁</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>q₂</td>
<td></td>
</tr>
</tbody>
</table>

and

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>q₁</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₃</td>
<td>a</td>
<td>c</td>
<td></td>
</tr>
</tbody>
</table>

et cetera.
More Legal Windows
The head can move in and out the window:

\[
\begin{array}{ccc}
  a & b & c \\
  a & b & q_2 \\
\end{array} \quad \begin{array}{ccc}
  b & a & q_4 \\
  b & a & c \\
\end{array} \quad \begin{array}{ccc}
  b & a & c \\
  q_7 & a & c \\
\end{array}
\]
et cetera.

The tape and the TM remain stationary once an accepting state has been reached:

\[
\begin{array}{ccc}
  a & q_{\text{accept}} & b \\
  a & q_{\text{accept}} & b \\
\end{array}
\]
Some Illegal Windows
For various reasons the following windows indicate a mistake in the configuration sequence:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>b</th>
<th>a</th>
<th>q₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>q₂</td>
<td>q₁</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>q₂</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₆</td>
<td>a</td>
<td>c</td>
</tr>
</tbody>
</table>

Most important, this window: is illegal if \((q₉,c,R) \notin \delta(q₄,b)\)
Let $L \subset (Q \cup \Gamma \cup \{\#\})^6$ be the set of legal windows. Then “(i,j) window legal” can be expressed by

$$\bigcup_{(s_1, \ldots, s_6) \in L} (x_{(i,j,s_1)} \land x_{(i,j+1,s_2)} \land \cdots \land x_{(i+1,j+2,s_6)})$$

With these sub-formulas we can express

$$\phi_{move} = \bigcap_{i=1}^{n^k-1} \bigcap_{j=1}^{n^k-2} (i, j) \text{ window legal}$$
The Complete $\phi_w$ Formula

Together these four requirements give:

$$\phi_w = \phi_{\text{cell}} \land \phi_{\text{start}} \land \phi_{\text{move}} \land \phi_{\text{accept}}$$

By construction, $\phi_w$ is satisfiable if and only if there is an accepting nondeterministic path of $N$ on input $w$:

$$w \in A \iff \phi_w \in \text{SAT}$$

We have to check that $w \rightarrow \phi_w$ is poly-time…
Polytime Reduction Check 1
Fixed TM N with time complexity $O(n^k)$ on input $w_1,\ldots,w_n$ of length $n$.

There are $O(n^{2k})$ Boolean variables $x_{(i,j,s)}$ in $\phi_w$.

Last issue: Can we describe $\phi_w$ in $\text{poly}(n)$ time?

$$\phi_{\text{cell}} = \bigcap_{i,j=1}^{n^k} \text{ONE}(x_{(i,j,1)},\ldots,x_{(i,j,c)})$$

$O(n^{2k})$ time.
Can we describe $\phi$ in poly(n) time?

$$\phi_{\text{start}} = x_{(1,1,#)} \land x_{(1,2,q_0)} \land$$

$$x_{(1,3,w_1)} \land \cdots \land x_{(1,n+2,w_n)} \land$$

$$x_{(1,n+3,-)} \land \cdots \land x_{(1,n^k-1,-)} \land x_{(1,n^k,#)}$$

… requires $O(n^k)$ time.
Polytime Reduction Check 3

Can we describe $\phi$ in poly(n) time?

\[
\phi_{\text{move}} = \bigcap_{i=1}^{n^k-1} \bigcap_{j=1}^{n^k-2} (i, j) \text{ window legal}
\]

... requires $O(n^{2k})$ time.

\[
\phi_{\text{accept}} = \bigcup_{j=1}^{n^k} x_{(n^k, j, q_{\text{accept}})}
\]

... complexity $O(n^k)$. 

Given $w_1, \ldots, w_n$, the construction of

$$\phi_w = \phi_{\text{cell}} \land \phi_{\text{start}} \land \phi_{\text{move}} \land \phi_{\text{accept}}$$

requires $O(n^{2k})$ time and space: poly(n).

The mapping from $w_1, \ldots, w_n$ to $\phi_w$ is a poly-time reduction from the \textbf{NP} problem A to SAT: $A \leq_p \text{SAT}$ for all $A \in \textbf{NP}$.

As SAT $\in \textbf{NP}$, we see that SAT is \textbf{NP}-complete.
3SAT is NP-Complete As Well

Corollary 7.32 describes how the previous reduction can also be done to a 3CNF formula $\varphi_w$.

First we want to transform $\varphi_w$ into CNF. Only the $(i,j)$-legal window parts are not yet in CNF:

$$\bigcup_{(s_1,\ldots,s_6) \in L} (x_{(i,j,s_1)} \land x_{(i,j+1,s_2)} \land \cdots \land x_{(i+1,j+2,s_6)})$$

We can rewrite these using the distributive law: $P \lor (Q \land R) = (P \lor Q) \land (P \lor R)$. This will give only a constant blow-up of $\varphi_w$. 
From CNF to 3NF

Here is how to make a general CNF formula into a 3CNF formula without changing the fact whether it is satisfiable or not:

Map the $m$ literals of the clause $(a_1 \lor a_2 \lor \cdots a_m)$ to the $m-2$ clauses

$$
(a_1 \lor a_2 \lor z_1) \land (\bar{z}_1 \lor a_3 \lor z_2) \land (\bar{z}_2 \lor a_4 \lor z_3) \land \cdots

\cdots \land (\bar{z}_{m-3} \lor a_{m-1} \lor a_m)
$$

Again the length of the new 3CNF formula is only a constant times the length of the old CNF one.
3SAT is NP-Complete As Well

Proof Corollary 7.32: 3SAT is $\textbf{NP}$-complete. Let $A$ be an $\textbf{NP}$ language with $O(n^k)$ NTM $N$. Given input $w_1, \ldots, w_n$, we make the Boolean formula $\phi_w = \phi_{\text{cell}} \land \phi_{\text{start}} \land \phi_{\text{move}} \land \phi_{\text{accept}}$ in time $O(n^{2k})$. From this $\phi_w$ we make the 3CNF formula $\varphi_w$ in time $O(|\phi_w|) = O(n^{2k})$. By construction: $w \in A \iff \varphi_w \in \text{3SAT}$.

Because the language 3SAT is in $\textbf{NP}$, it follows that 3SAT is $\textbf{NP}$-complete.
Proving NP-Completeness

First prove that the language A is in \textbf{NP}.

Then, reduce one of the known \textbf{NP}-complete problems to A.
(Give a reference for the \textbf{NP}-complete problem.)

At the moment we know that SAT, 3SAT and CLIQUE are \textbf{NP}-complete.

When proving that A is in \textbf{P}, just give a poly-time algorithm.
Examples SUBSET-SUM

\[ \langle \{2,4,8\}, 10 \rangle \in \text{SUBSET-SUM} \quad \text{... because } 2+8=10 \]

\[ \langle \{2,4,8\}, 11 \rangle \notin \text{SUBSET-SUM} \quad \text{... because } 11 \text{ cannot be made out of } \{2,4,8\} \]

\[
\text{SUBSET-SUM} = \{ \langle S, t \rangle \mid S = \{x_1, \ldots, x_k\} \quad \text{there is a subset } R \subseteq S \quad \text{such that } \sum_{y \in R} y = t \}\]
Reducing 3SAT to SubSet Sum

**Proof idea:**

- Choosing the subset numbers from the set S corresponds to choosing the assignments of the variables in the 3CNF formula.
- The different digits of the sum correspond to the different clauses of the formula.
- If the target t is reached, a valid and satisfying assignment is found.
Subset Sum

3CNF formula:

\[(x_1 \lor x_2 \lor x_3) \land (\overline{x}_1 \lor x_2 \lor x_4) \land (\overline{x}_2 \lor \overline{x}_2 \lor \overline{x}_3) \land (x_1 \lor \overline{x}_3 \lor \overline{x}_4)\]

Make the number table, and the ‘target sum’ t
Reducing 3SAT to SubSet Sum

- Let $\phi \in 3\text{CNF}$ with $k$ clauses and $\ell$ variables $x_1, \ldots, x_\ell$.

- Create a Subset-Sum instance $<S_\phi, t>$ by:
  
  - $2\ell + 2k$ elements of $S_\phi = \{y_1, z_1, \ldots, y_\ell, z_\ell, g_1, h_1, \ldots, g_k, h_k\}$
  
    - $y_j$ indicates positive $x_j$ literals in clauses
    - $z_j$ indicates negated $x_j$ literals in clauses
    - $g_j$ and $h_j$ are dummies
    - and

  $$t = \underbrace{1 \ldots 1}_{\ell} \underbrace{3 \ldots 3}_{k}$$
Note 1: The “1111” in the target forces a proper assignment of the $x_i$ variables.

Note 2: The target “3333” is only possible if each clause is satisfied. (The dummies can add maximally 2 extra.)
Subset Sum

Let's consider a truth table for the expressions $x_1, \bar{x}_2, \bar{x}_3, x_4$. Each row represents a possible assignment of truth values to these variables, and the columns show the results of applying the logical operators $\lor$, $\land$, and $\lor$.

We are looking for a satisfying assignment, which means there exists an assignment of variables such that the result is true. In this case, the assignment $x_1, \bar{x}_2, \bar{x}_3, x_4$ results in a truth value of 1 in the rightmost column, indicating that this is a satisfying assignment.

The truth table confirms that for the given assignment of variables, the overall expression evaluates to true.
Subset Sum

\[ \bar{x}_1, \bar{x}_2, \bar{x}_3, x_4 \text{ is not a satisfying assignment} \]
Proof $3\text{SAT} \leq_p \text{Subset Sum}$

- For every $3\text{CNF } \phi$, take target $t=1\ldots13\ldots3$ and the corresponding set $S_\phi$.

- If $\phi \in 3\text{SAT}$, then the satisfying assignment defines a subset that reaches the target.

- Also, the target can only be obtained via a set that gives a satisfying assignment for $\phi$.

\[ \phi \in 3\text{SAT} \text{ if and only if } \left( S_\phi, 1\ldots13\ldots3 \right) \in \text{Subset - Sum} \]
Finding the Solution

- If the **decision problem** Subset-Sum is solvable in polynomial time, then we can also find the subset that reaches the target in poly-time.

- How?

- By asking smart questions about several variants of the problem.

- That way, we can determine which variables $x_i$ are involved in the solution.
Self Reducibility

- Take Subset-Sum instance $S=\{x_1, \ldots, x_k\}$, $t$
  
  **Question**: “Which $x$-terms sum up to $t$?”

- Solve “$({x_2, \ldots, x_k}, t) \in \text{Subset-Sum}$?”
  - If “Yes”, then $x_1$ is not crucial for the solution
  - If “No”, then $x_1$ is part of the $t$-sum

- Solve similar questions for $x_2, \ldots, x_k$
Directed Hamiltonian Path

- Given a directed graph $G$, does there exist a Hamiltonian path, which visits all nodes exactly once?

- Two Examples:
  - This graph has a Hamiltonian path
  - But this graph has no Hamiltonian path

![Diagrams showing examples of graphs with and without Hamiltonian paths](image-url)
3SAT to Hamiltonian Path

- **Proof idea:**
  Given a 3CNF $\phi$, make a graph $G_\phi$ such that

  - a Hamiltonian path is possible if and only if there is a zig-zag path through $G_\phi$,
  - where the directions (zig or zag) determine the assignments (true or false) of the variables $x_1, \ldots, x_L$ of $\phi$.  

"Zig-Zag Graphs"

There are $2^L$ different Hamiltonian paths from $s$ to the target $t$.

For every $i$, "Zig" means $x_i=True$

...while "Zag" means $x_i=False$
If $\phi$ has $K$ clauses, $\phi = C_1 \land \ldots \land C_k$, then $x_i$ has $K$ “hubs”:

**Idea**: Make $K$ extra nodes $C_j$ that can only be visited via the hub-path that defines an assignment of $x_i$ which satisfies the clause (using zig/zag = T/F).
Connecting the Clauses

Let the clause $C_j$ contain $x_i$:

If $C_j = (x_i \lor y \lor z)$
then $x_i = \text{True} = \text{"zig"}$
reaches node $C_j$

If $C_j = (\overline{x_i} \lor y \lor z)$
then $x_i = \text{False} = \text{"zag"}$
reaches node $C_j$
Proof $3\text{SAT} \leq_p \text{HamPath}$

- Given a 3CNF $\phi(x_1, \ldots, x_L)$ with $K$ clauses
- Make graph $G_\phi$ with a zig/zag levels for every $x_i$
- Connect the clauses $C_j$ via the proper “hubs”

- If $\phi \in \text{SAT}$, then the satisfying assignment defines a Hamiltonian path, and vice versa.

$\phi \in \text{SAT}$ if and only if $G_\phi \in \text{HamPath}$
Example

4 variables...

4 clauses...

Clauses connected via zig-zag “hubs”

\[(x_1 \lor x_2 \lor x_3) \land (\overline{x}_1 \lor x_2 \lor x_4) \land (\overline{x}_2 \lor \overline{x}_2 \lor \overline{x}_3) \land (x_1 \lor \overline{x}_3 \lor \overline{x}_4)\]
Example

\[ x_1, \overline{x}_2, \overline{x}_3, x_4 \]

assignment...

...satisfies all four clauses; hence it defines a Hamiltonian Path

\[(x_1 \lor x_2 \lor x_3) \land (\overline{x}_1 \lor x_2 \lor x_4) \land (\overline{x}_2 \lor \overline{x}_2 \lor \overline{x}_3) \land (x_1 \lor \overline{x}_3 \lor \overline{x}_4)\]
Example

\( \overline{x}_1, \overline{x}_2, \overline{x}_3, x_4 \) assignment...

...does not satisfy first clause; hence the path misses the \( C_1 \) node

\[
(x_1 \lor x_2 \lor x_3) \land (\overline{x}_1 \lor x_2 \lor x_4) \land \\
(\overline{x}_2 \lor \overline{x}_2 \lor \overline{x}_3) \land (x_1 \lor \overline{x}_3 \lor \overline{x}_4)
\]
More on Hamiltonian Path

- Useful for proving NP-completeness of “optimal routing problems”

- Typical example: NP-completeness of “Traveling Salesman Problem”

- Issue of directed versus undirected graphs
Forcing Directions

Given a directed graph with $s, x_1, \ldots, x_k, t$

Replace $s$ with $s_{out}$, the $t$ with $t_{in}$, and every $x_i$ with the triplet "$x_i.in \rightarrow x_i.mid \rightarrow x_i.out$"

Redraw the original directed edges with edges going from out-nodes to in-nodes.

If and only if the directed graph has a HamPath from $s$ to $t$, then so has this graph.
Example: Directions

becomes

becomes

“Undirected HamPath” is NP-complete
Minimizing Miles

- Given k cities, with distance graph G
- “Is there a route that visits all k cities with less than t miles of traveling?”

- The Traveling Salesman Problem is in NP.
- Close connection with Undirected-Ham-Path
- One can show: TSP is NP-complete
Conclusions

Time complexity

P Class

NP Class

P vs. NP

NP-Completeness