



Complexity IBC028, Lecture 6

Niels van der Weide

Institute for Computing and Information Sciences
Radboud University Nijmegen

Version: spring 2025

Outline

Three more NP-complete problems

Extra Topics

PSPACE



How to prove that a problem is **NP**-complete?

- ① Prove that $A \in \mathbf{NP}$: give a polynomial algorithm f such that f verifies A with polynomial certificates, that is:

$$x \in A \iff \exists y \in \{0, 1\}^* (|y| \text{ polynomial in } |x| \wedge f(x, y) = 1)$$
- ② Pick a well-known decision problem B which you know is **NP**-hard,
- ③ Prove that $B \leq_P A$, that is:
 give a polynomial function h such that

$$x \in B \iff h(x) \in A.$$

Some **NP**-complete problems (satisfiability)

SAT

- Given a formula φ , is φ satisfiable?

That is: is there an assignment v such that $v(\varphi) = 1$?

CNF

- Given a formula φ in conjunctive normal form, is φ satisfiable?

$\leq_3$ CNF

- Given a formula in “at most 3-conjunctive normal form”, is it satisfiable?

3CNF

- Given a formula in “exactly 3-conjunctive normal form”, is it satisfiable?

Some NP-complete problems (integers)

ILP

- Given an integer linear program, does it have a solution?
For example

$$E := \begin{cases} x_1 + 3x_2 - 4x_3 + x_4 & \geq 5 \\ 3x_1 + x_2 + 4x_3 + 2x_4 & \leq 6 \\ 3x_1 - 2x_2 - x_3 - 3x_4 & \geq 0 \end{cases}$$

Are there $x_1, x_2, x_3, x_4 \in \mathbb{Z}$ such that these inequalities hold?

Some NP-complete problems (graphs)

Clique:

- Given a graph $G = (V, E)$ and an integer k , does G have a clique with k vertices?

That is: is there a set $W \subseteq V$ of size k with an edge between each pair of vertices ?

VertexCover

- Given a graph $G = (V, E)$ and an integer k , does G have a vertex cover with k vertices?

That is: is there a set $W \subseteq V$ of size k such that each edge “lands in” a vertex in W ?

3Color

- Given a graph $G = (V, E)$, does G have a 3-coloring?

That is: is there a function $c : V \rightarrow \{r, y, b\}$ such that $(v, u) \in E \Rightarrow c(v) \neq c(u)$.

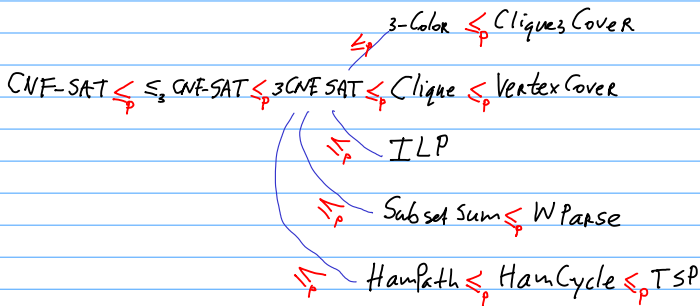
How to prove that a problem is **NP**-complete?

- In our proofs of **NP**-hardness we have used the following chain of reductions of satisfiability problems.

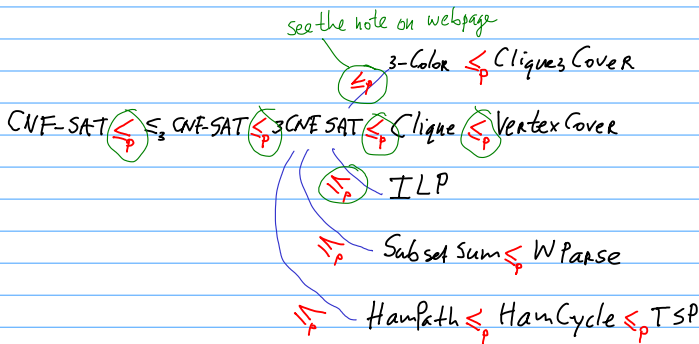
$$\text{CNF-SAT} \leq_P \leq_3 \text{CNF-SAT} \leq_P 3\text{CNF-SAT}$$

- We have extended this with proofs of **NP**-hardness of ILP, Clique, VertexCover and 3Color.
- For a proof of **NP**-hardness of Ham-Path (Hamiltonian path), see the note of Niels van der Weide on the webpage, by a reduction: $3\text{CNF-SAT} \leq_P \text{Ham-Path}$.
- In this lecture, we will prove **NP**-hardness of Clique-3Cover, SubsSum (Subset-Sum) WParse (weighted parsing), Ham-Cycle and TSP(traveling salesperson).

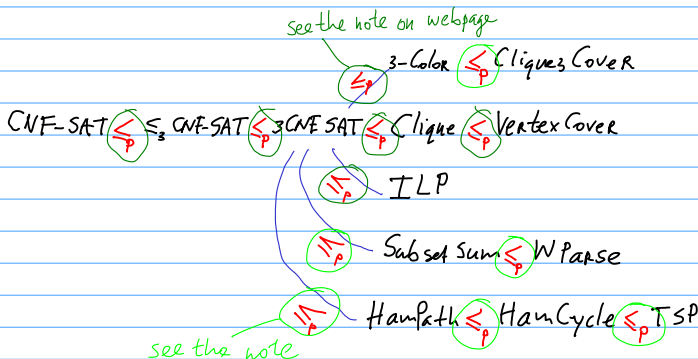
A hierarchy of NP-completeness proofs



A hierarchy of NP-completeness proofs: last week



A hierarchy of NP-completeness proofs: this week



Clique-3Cover is NP-complete

DEFINITION

Clique-3Cover is the problem of deciding if a graph $G = (V, E)$ is the union of three cliques, that is: $\exists V_1, V_2, V_3 (V = V_1 \cup V_2 \cup V_3 \wedge V_1 \cap V_2 = \emptyset, V_2 \cap V_3 = \emptyset, V_1 \cap V_3 = \emptyset \wedge \forall i (V_i \text{ is a clique}))$.

THEOREM

Clique-3Cover is **NP**-complete

- Clique-3Cover $\in$ **NP**. The sets (V_1, V_2, V_3) are a certificate.
- We show that **3Color** $\leq_P$ **Clique-3Cover** by defining $f(V, E) := (V, \bar{E})$, where $\bar{E} := \{(u, v) \mid u \neq v \wedge (u, v) \notin E\}$.
- (V, E) is 3-colorable iff $(V, \bar{E})$ has a clique-3cover, because

$$\begin{aligned} V_i \text{ is a clique in } (V, \bar{E}) &\Leftrightarrow \forall u, v \in V_i (u \neq v \rightarrow (u, v) \in \bar{E}) \\ &\Leftrightarrow \forall u, v \in V_i (u = v \vee (u, v) \notin E) \\ &\Leftrightarrow V_i \text{ can have one color in } (V, E). \end{aligned}$$

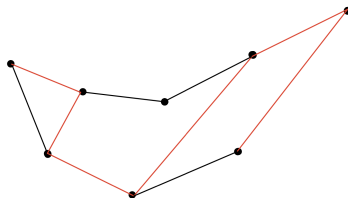
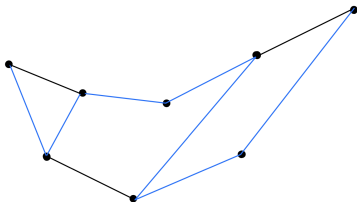
Hamiltonian paths

Definition

Let G be a graph. We say that G has a **Hamiltonian path** if there is a path p in G that crosses every vertex exactly once.

$$\text{Ham-Path} := \{(V, E) \mid \exists v_1, \dots, v_n (V = \{v_1, \dots, v_n\} \wedge \\ \forall i, j \leq n (v_i = v_j \rightarrow i = j) \wedge \\ \forall i < n (v_i, v_{i+1}) \in E)\}$$

Below, the blue path is Hamiltonian while the red is not.



NP-completeness

We look at the decision problem Ham-Path

- Given a graph G , does G have a Hamiltonian path?

Theorem

Ham-Path is **NP**-complete

It can be shown that $3\text{CNF-SAT} \leq_P \text{Ham-Path}$, See note!

Hamiltonian cycle

DEFINITION

Let G be a graph. We say that G has a **Hamiltonian cycle** if there is a **cycle** c in G that crosses every vertex exactly once.

$$\text{Ham-Cycle} := \{(V, E) \mid \exists v_1, \dots, v_n (V = \{v_1, \dots, v_n\} \wedge \\ \forall i, j < n (v_i = v_j \rightarrow i = j) \wedge \\ v_n = v_1 \wedge \forall i < n (v_i, v_{i+1}) \in E)\}$$

So, a cycle c is written as $v_1, v_2, \dots, v_n$ such that $(v_i, v_{i+1}) \in E$ and $(v_n, v_1) \in E$. For a Hamiltonian cycle: $v_i \neq v_j$ if $i \neq j$ and every vertex occurs in this cycle.

Ham-Cycle is **NP**-complete

Theorem

Ham-Cycle is **NP**-complete

Clearly, Ham-Cycle $\in$ **NP**.

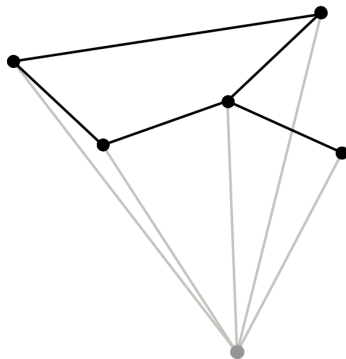
To prove that Ham-Cycle is **NP**-hard, we show

Ham-Path $\leq_P$ **Ham-Cycle**.

- Let $G = (V, E)$ be a graph;
- Add a vertex v and connect it to every vertex $u \in V$ an edge;
- Call the resulting graph G' ;
- **LEMMA**
 G has a Hamiltonian path iff G' has a Hamiltonian cycle

Illustration of the proof

From the construction, we get the following graph



Traveling Salesperson, TSP

DEFINITION

Given a **complete** graph G , a cost function on the edges c , and an integer k , is there a cycle in G with cost at most k that crosses every vertex?

TSP := $\{(V, c, k) \mid c : V \times V \rightarrow \mathbb{Z} \wedge k \in \mathbb{Z} \wedge \text{there is a cycle crossing all vertices with cost at most } k\}$

Theorem

TSP is **NP-complete**.

PROOF

- TSP $\in$ **NP**.

The certificate is the cycle (the “tour” of the TSP). That it has cost $\leq k$ can be checked easily in polynomial time.

TSP is NP-hard

- TSP $\in$ **NPH**. We show **Ham-Cycle** $\leq_P$ **TSP**.

Define for (V, E) a graph the following tuple (V, c, k) , consisting of a complete graph, a $c : V \times V \rightarrow \mathbb{Z}$, $k \in \mathbb{Z}$.

- $c(u, v) := 1$ if $(u, v) \in E$,
 $c(u, v) := 2$ if $(u, v) \notin E$
- $k := |V|$

LEMMA (V, E) has a Hamiltonian cycle if and only if (V, c) has a tour with cost at most $|V|$

PROOF

Check $\Rightarrow$ and $\Leftarrow$.

COROLLARY **Ham-Cycle** $\leq_P$ **TSP** and so: TSP is **NP**-hard.

SubsSum, the subset-sum problem

DEFINITION

SubsSum(S, t) is the problem of deciding, for $S \subseteq_{\text{fin}} \mathbb{N}$ and $t \in \mathbb{N}$, if there is a subset $S' \subseteq S$ such that $\sum_{x \in S'} x = t$. Here, $S \subseteq_{\text{fin}} \mathbb{N}$ denotes that S is a **finite** subset of $\mathbb{N}$.

Example: take $S = \{1, 4, 6, 9, 12\}$

- There is a subset with sum 14, namely $\{1, 4, 9\}$
- There is no subset with sum 8

We assume the representation of a number $n \in \mathbb{N}$ to be of size $\Theta(\log n)$. This holds for binary or decimal (but for not unary!). For simplicity we now assume **decimal** representation.



SubsSum is NP-complete

THEOREM

SubsSum is **NP**-complete

- SubsSum $\in$ **NP**.

The certificate is the subset $S' \subseteq S$ whose sum is t .

- We prove SubsSum is **NP-hard** by showing $\leq_3\text{CNF-SAT} \leq_P \text{SubsSum}$.

We define $f : \leq_3\text{CNF} \rightarrow \mathcal{P}_{\text{fin}}(\mathbb{N}) \times \mathbb{N}$ such that

$\varphi = \bigwedge_{i=1}^k C_i$ is satisfiable if and only if

for $f(\varphi) = (S, t)$ there is a $S' \subseteq S$ with $\sum_{x \in S'} x = t$.

SubsSum is NP-hard

For the definition of $f : \leq_3\text{CNF} \rightarrow \mathcal{P}_{\text{fin}}(\mathbb{N}) \times \mathbb{N}$:

- Assume that $\varphi = \bigwedge_{i=1}^k C_i$ has n atoms $\{x_1, \dots, x_n\}$.
 S will consist of numbers of $n + k$ digits and t will also have $n + k$ digits.
- Define numbers $p_1, p'_1, \dots, p_n, p'_n$ by:
 - p_i has: 1 at position i and 1 at pos. $n + j$ if x_i occurs in C_j ,
 - p'_i has: 1 at position i and 1 at pos. $n + j$ if $\neg x_i$ occurs in C_j ,
 - all other positions in p_i and p'_i are 0.
- Define numbers $s_1, s'_1, \dots, s_k, s'_k$ by:
 - s_j has 1 at position $n + j$ and for the rest 0,
 - s'_j has 2 at position $n + j$ and for the rest 0.
- Take $S = \{p_i, p'_i \mid i = 1, \dots, n\} \cup \{s_j, s'_j \mid j = 1, \dots, k\}$ and $t = 1 \dots 14 \dots 4$ (n times a 1 and k times a 4).
- LEMMA: φ is satisfiable iff $\exists S' \subseteq S (\sum_{x \in S'} x = t)$.

$\leq_3$ CNF-SAT $\leq_P$ SubSum: Example

- p_i has 1 at position i and at position $n + j$ if x_i occurs in C_j ,
- p'_i has 1 at position i and at position $n + j$ if $\neg x_i$ occurs in C_j .

						n (=3)			k(=4)				
(C ₁)	x ₁	∨	¬x ₂	∨	¬x ₃	p ₁	1	0	0	1	0	0	1
(C ₂)			¬x ₂	∨	x ₃	p' ₁	1	0	0	0	0	1	0
(C ₃)	¬x ₁	∨	x ₂			p ₂	0	1	0	0	0	1	0
(C ₄)	x ₁	∨	¬x ₂	∨	¬x ₃	p' ₂	0	1	0	1	1	0	1
						p ₃	0	0	1	0	1	0	0
						p' ₃	0	0	1	1	0	0	1

- Basically, the first n columns represent the atoms $x_1, \dots, x_n$ and the last k columns represent the clauses $C_1, \dots, C_k$.
- Using a satisfying assignment v for φ , we choose p_i or p'_i for each i (depending on $v(x_i) = 1 / 0$).
- Summing up these p 's we get $t' = 1 \dots 1 d_1 \dots d_k$ with $d_j \in \{1, 2, 3\}$, because ≥ 1 literal in each clause is true.
- So we can add specific s_j and s'_j to sum up to $t = 1 \dots 14 \dots 4$

Parsing and Weighted parsing

- Given a Context Free Grammar (CFG) G and a word w , can we derive **Start** $\Rightarrow w$?
- This is the Parse-problem.
- Put differently: Is there a **parse-tree** for w ?
- The Parse problem can be solved in polynomial time. (E.g. CYK-algorithm)

Variant of the problem WParse, is there a **weighted parse tree** for w of weight k ?

DEFINITION

Given a CFG G where every production rule has a **weight**, let **Start** $\xRightarrow{m} w$ denote that w has a parse tree where the sum of the weights of all production rules is m .

WParse(G, w, k) is the problem **Start** $\xRightarrow{k} w$: Is there a parse tree of w with weight k ?

Example: parsing and weighted parsing

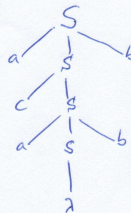
Example

$$S \rightarrow a S b$$

$$S \rightarrow c S$$

$$S \rightarrow \lambda$$

$$S \Rightarrow a c a b b$$



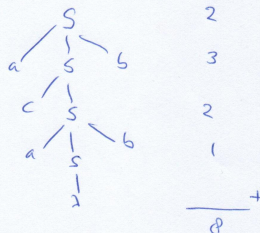
Example

$$S \xrightarrow{2} a S b$$

$$S \xrightarrow{3} c S$$

$$S \xrightarrow{1} \lambda$$

$$S \xRightarrow{8} a c a b b$$



WParse is NP-complete

THEOREM

WParse is **NP**-complete

Proof.

- ① WParse $\in$ **NP**.

The certificate is the parse tree of w with weight k

- ② We show that WParse is **NP**-hard by showing

SubsSum $\leq_P$ **WParse**.

Given $S = \{s_1, \dots, s_n\}$ and $k \in \mathbb{N}$ define the following weighted grammar:

Start $\xrightarrow{0} A_1 \dots A_n$, $A_i \xrightarrow{0} B_i$, $A_i \xrightarrow{0} \lambda$, $B_i \xrightarrow{s_i} \lambda$.

Then

$$\exists S' \subseteq S (\sum S' = k) \quad \text{iff} \quad \text{Start} \xrightarrow{k} \lambda.$$

Decision problems versus optimization problems

VertexCover

- Given a graph G and an integer k , does G have a vertex cover with k vertices?
- Given a graph G , find the **minimal** vertex cover of G .

TSP

- Given a complete graph G , a function c , and an integer k , is there a cycle in G with cost at most k ?
- Given a complete graph G and a function c , find a cycle in G with **minimal** cost.

What does **NP**-completeness mean for optimization problems?

- Since we know VertexCover and TSP are **NP**-complete, it will be either difficult or impossible to find efficient algorithms that compute minimal vertex covers or minimal cycles.
- More precisely: If a minimal solution can be computed in polynomial time, then also the decision problem is in **P**.
- But what if we do not go for the **best** solution, but instead for a solution that is **good enough**?

Example: Finding vertex covers (I)

Let $G = (V, E)$ be a graph.

To compute a vertex cover C of G , we take the following steps

- Let $C := \emptyset$ and $E' := E$
- While E' is not empty
 - ① Take any edge (u, v) from E'
 - ② Take $C := C \cup \{u, v\}$
 - ③ Remove every edge from E' that touches either u or v .
- output: C

Example: Finding vertex covers (II)

The algorithm is polynomial; it doesn't find the minimal vertex cover...but it is a “decent approximation”.

Theorem

The size of the vertex cover computed by the algorithm, is at most twice as large as the size of the minimal vertex cover.

So: while the algorithm does not give the best solution, it gives a solution within reasonable time that may be “good enough” for our purpose.

SAT-solvers

Even though SAT is **NP**-complete, there are very powerful tools that decide whether a (very large) formula is satisfiable. These are called **SAT-solvers**.

These have been (and are continuously further) optimized and can now deal with tens of thousands of variables and millions of clauses.

SAT-solvers are the “automation workhorses” of computer science.

Example: negative solution to the “Boolean Pythagorean triples problem”.

See: Master Course Automated Theorem Proving.

Harder than NP

- There are problems that don't have a polynomial checking algorithm, or for which the certificate is not polynomial.
- Example: Two-player games.
 - “Is there a winning strategy for player 1?”
 - Certificate is typically not polynomial size.

Next natural level after **P** (and **NP**): decision algorithms that are polynomially bound on **space** (memory use), not on time.

DEFINITION

f is a **polynomial space** algorithm for A if

- f is a deterministic Turing Machine that
- halts on every input w such that
- $w \in A$ iff $f(w)$ halts in a final state and
- the **size of the tape** used in the computation of $f(w)$ is polynomial in $|w|$.

PSPACE

PSPACE :=

$\{A \subseteq \{0, 1\}^* \mid \exists f, f \text{ polynomial space algorithm,}$
 $w \in A \iff f(w) = 1\}$

LEMMA

- **$P \subseteq \text{PSPACE}$**

Because in polynomial size time, f uses only polynomial size space.

- **$\text{NP} \subseteq \text{PSPACE}$**

Because if $A = \{w \mid \exists y(y < c|w|^k \wedge f(w, y) = 1)\}$, this can be checked using polynomial size space, by summing up all (exponentially many!) candidate y 's and running $f(w, y)$.

NPSPACE

Just like **NP**, we also have **NPSPACE**.

DEFINITION

f is a **non-deterministic polynomial space** algorithm for A if

- f is a **non-deterministic** Turing Machine that
- halts on every input w such that
- $w \in A$ iff $f(w)$ **has a computation** that halts in a final state,
- the **size of the tape** used in the computation of $f(w)$ is polynomial in $|w|$.

SAVITCH' THEOREM

$$\text{PSPACE} = \text{NPSPACE}$$

PSPACE-hard and PSPACE-complete

DEFINITION

- A is called **PSPACE-hard** if

$$\forall A' \in \mathbf{PSPACE} (A' \leq_P A).$$

(All **PSPACE**-problems can be **poly. time** reduced to A.)

- $\mathbf{PspaceH} := \{A \mid A \text{ is } \mathbf{PSPACE}\text{-hard}\}.$
- A is called **PSPACE-complete** if $A \in \mathbf{PSPACE}$ and A is **PSPACE-hard**.
- $\mathbf{PspaceC} := \mathbf{PSPACE} \cap \mathbf{PspaceH}.$

THEOREM

If $A' \leq_P A$ and $A' \in \mathbf{PspaceH}$, then $A \in \mathbf{PspaceH}$.

The proof is the same as for **NP-hard**.

How to prove that A is **PSPACE**-complete?

Just like SAT is the canonical **NP**-hard problem, there is a canonical **PSPACE**-hard problem: **QBF**.

DEFINITION

A **quantified boolean formula** (QBF) is a boolean formula where we can now also use **quantifiers** ($\forall$, $\exists$) **over boolean variables**.

QBF is the problem of deciding whether a closed quantified boolean formula φ is true.

QBF is **PSPACE**-complete

Example $\varphi = \forall x (\exists y (x \wedge y)) \vee (\exists z (\neg x \wedge \neg z))$

- For $x = 1$ we can choose $y = 1$ and for $x = 0$ we can choose $z = 0$.
- That is: for all values of x we can choose a case and a value for y (or z) that makes the boolean formula true.
- So φ is true.

THEOREM

QBF is **PSPACE**-complete.

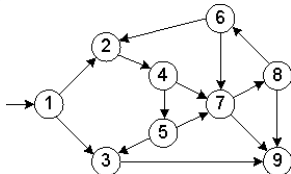
- The “certificate” for $\text{QBF}(\varphi)$ is not just a choice of 0 / 1 for every $\exists$, but a choice depending on the $\forall$ in front of the $\exists$.
- The proof that QBF is **PSPACE**-hard uses a translation of Turing Machines to QBF.

Some variations on QBF

- Note that $\text{SAT} \leq_P \text{QBF}$:
given φ add $\exists x$ in front of φ for all atoms x in φ .
- If we limit QBF to **prenex** formulas, that have all quantifiers in front, it is still **PSPACE**-complete.
- If we limit QBF to **alternating prenex** formulas, that have alternating $\forall/\exists$ in front, it is still **PSPACE**-complete.
- A “proof” of $\forall x_1 \exists y_1 \dots \forall x_n \exists y_n(\varphi)$ amounts to making n choices, which amounts to a “certificate” of size 2^n .
- A formula like $\forall x_1 \exists y_1 \dots \forall x_n \exists y_n(\varphi)$ can be interpreted as the question for a winning strategy for a two-player game.

Some other **PSPACE**-complete problems

- Strategic games are typically **PSPACE**-complete, like **Geography**



- Also **RushHouR** and **Sokoban** are **PSPACE**-complete.
- Given two regular expression e_1 and e_2 , do we have $\mathcal{L}(e_1) = \mathcal{L}(e_2)$? This problem is **PSPACE**-complete. Similarly: Equivalence problem for non-deterministic finite automata: Given two NFAs over Σ , do they accept the same language? (Note: for DFAs this problem is in **P**!)
- The word problem for **deterministic context-sensitive grammars** is **PSPACE**-complete. This is the problem whether **Start** $\Rightarrow w$ in such a grammar.