Complexity IBC028, Lecture 6

H. Geuvers

Institute for Computing and Information Sciences
Radboud University Nijmegen

Version: spring 2021

Outline

Proving that a problem is NP-complete

To prove that L is NP-complete, we proceed as follows.

1. Prove that $L \in$ NP: give a pol. algorithm and pol. certificate.
2. Pick a well-known $L' \in$ NPH (NP-hard) and show that $L' \leq_P L$.

- For showing NP-hardness we have used the following chain of satisfiability reductions.

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

- We have extended this with proofs of NP-hardness of ILP, Clique, VertexCover and 3Color.
- In the book you can find proofs of NP-hardness of Ham-Cycle(Hamiltonian cycle) and of SubsSum (Subset-Sum)
- In this lecture, we will prove NP-hardness of Clique-3Cover,WParse (weighted parsing) and TSP(traveling salesman).

A hierarchy NP-completeness proofs

Some polynomial Reductions to prove NP-hardness

3-Color $\leq_p$ Clique 3Cover

$\mathbin{\rotatebox{90}{$\leq$}}_p$

SAT $\leq_p$ CNF-SAT $\leq_p$ $\leq_3$CNF-SAT $\leq_p$ 3CNF-SAT $\leq_p$ ILP

$\Uparrow_p$

SubsetSum $\leq_p$ WParse

$\leq_p$ Clique $\leq_p$ VertexCover $\leq_p$ Hamcycle $\leq_p$ TSP

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 $f(V, E) := (V, \overline{E})$, where $\overline{E} := \{(u, v) \mid u \neq v \wedge (u, v) \notin E\}$.
- (V, E) is 3-colorable iff $(V, \overline{E})$ has a clique-3cover, because

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

SubsSum is NP-complete

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 $\Sigma S' = t$.
Here, $S \subseteq_{\text{fin}} \mathbb{N}$ denotes that S a finite subset of $\mathbb{N}$ and $\Sigma S'$ denotes the sum of all elements in S' (also: $\Sigma_{x \in S'} x$).

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 simplicty we now assume decimal representation.

Theorem

SubsSum is NP-complete

- SubsSum $\in$ NP. The certificate is the subset $S' \subseteq S$ whose sum is t.
- We can prove SubsSum is NP-hard by showing $\leq_3$CNF-SAT $\leq_P$ SubsSum.

SubsSum is NP-hard ($\leq_3$CNF-SAT $\leq_P$ SubsSum).

We define $f : \leq_3\text{CNF} \to \mathcal{P}_{\text{fin}}(\mathbb{N}) \times \mathbb{N}$ such that $\varphi = \bigwedge_{i=1}^{k} C_i$ is satisfiable iff for $f(\varphi) = (S, t)$ there is a $S' \subseteq S$ with $\Sigma S' = t$.

- Assume that $\varphi = \bigwedge_{i=1}^{k} C_i$ has n atoms $\{x_1, \ldots, x_n\}$.
- Define numbers $p_1, p_1', \ldots, p_n, p_n'$ (each with $n + k$ digits) 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 $s_1, s_1', \ldots, s_k, s_k'$ (each with $n + k$ digits) 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 $S = \{p_i, p_i' \mid i = 1, \ldots, n\} \cup \{s_j, s_j' \mid j = 1, \ldots, k\}$ and $t = 1 \ldots 14 \ldots 4$ (n times a 1 and k times a 4).
- LEMMA: φ is satisfiable iff $\exists S' \subseteq S(\Sigma S' = t)$.

$\leq_3$CNF-SAT $\leq_P$ SubsSum: 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	$\vee$	$\neg x_2$	$\vee$	$\neg x_3$	p_1	1	0	0	1	0	0	1
(C_2)			$\neg x_2$	$\vee$	x_3	p'_1	1	0	0	0	0	1	0
(C_3)	$\neg x_1$	$\vee$	x_2			p_2	0	1	0	0	0	1	0
(C_4)	x_1	$\vee$	$\neg x_2$	$\vee$	$\neg x_3$	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 colums represent the atoms $x_1, \ldots, x_n$ and the last k colums represent the clauses $C_1, \ldots, C_k$.
- 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 $t' = 1 \ldots 1 d_1 \ldots d_k$ 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 $t = 1 \ldots 1 4 \ldots 4$

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 $\stackrel{m}{\Rightarrow} w$ 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 $\stackrel{k}{\Rightarrow} w$: Is there a parse tree of w with weight k?

Example: parsing and weighted parsing

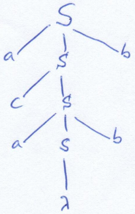
Example

$S \to aSb$

$S \to cS$

$S \to \lambda$

$S \Rightarrow acabb$



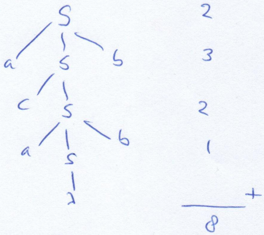
Example

$S \xrightarrow{2} aSb$

$S \xrightarrow{3} cS$

$S \xrightarrow{1} \lambda$

$S \overset{8}{\Rightarrow} acabb$



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 $S = \{s_1, \ldots, s_n\}$ and $k \in \mathbb{N}$ define the following weighted grammar:
   $\text{Start} \xrightarrow{0} A_1 \ldots A_n, \quad A_i \xrightarrow{0} B_i, \quad A_i \xrightarrow{0} \lambda, \quad B_i \xrightarrow{s_i} \lambda.$
   Then
   $$\exists S' \subseteq S(\Sigma S' = k) \qquad \text{iff} \qquad \text{Start} \overset{k}{\Rightarrow} \lambda.$$

Ham-Cycle and TSP-complete

Ham-Cycle is the set of graphs containing a Hamiltonian cycle:

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

A variation is Ham-Path the set of graphs containing a Hamiltonian path. Then we drop the $v_n = v_1$ requirement.

Ham-Cycle is NP-complete because (1) it is in NP (check!) and (2) it is NP-hard, because it can be shown (see the book in case you want to see the details) that VertexCover $\leq_P$ Ham-Cycle.

$$\begin{aligned}\text{TSP} := \{(V,E,c,k) \mid\ & (V,E) \text{ is complete} \wedge c : V \times V \rightarrow \mathbb{Z} \wedge \\ & k \in \mathbb{Z} \ \wedge \ \text{there is a cycle with cost at most } k\}\end{aligned}$$

THEOREM TSP is NP-complete.

PROOF

- TSP $\in$ NP. The certificate is the cycle; That it has cost $\leq k$ can be checked easily .

TSP is NP-hard

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

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

  - $E' = V \times V$
  - $c(u, v) := 0$ if $(u, v) \in E$,
    $c(u, v) := 1$ if $(u, v) \notin E$
  - $k := 0$

  LEMMA (V, E) has a Hamiltonian cycle if and only if (V, E') has a tour with cost at most 0

  PROOF
  Check $\Rightarrow$ and $\Leftarrow$.

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

Harder then 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: decision algorithms that are polynomially bound on **space** (memory use), not on time.

DEFINITION

A is a polynomial space algorithm for L if

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

PSPACE

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

Lemma

- P $\subseteq$ PSPACE
  Because in polynomial size time, A uses only polynomial size space.
- NP $\subseteq$ PSPACE
  Because if $L = \{w \mid \exists y (y < c|w|^k \wedge A(w,y) = 1\}$, this can be checked using polynomial size space, by summing up all (exponentially many!) candidate y's and running $A(w,y)$.

NPSPACE

Just like NP, we also have NPSPACE.

DEFINITION

A is a non-deterministic polynomial space algorithm for L if

- A is a non-deterministic Turing Machine that
- halts on every input w such that
- $w \in L$ iff $A(w)$ has a computation that halts in q_f and
- the size of the tape used in the computation of $A(w)$ is polynomial in $|w|$.

SAVITCH' THEOREM

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

PSPACE-hard and PSPACE-complete

Definition

- L is called PSPACE-hard if
$$\forall L' \in \text{PSPACE}(L' \leq_P L).$$
That is: all PSPACE-problems can be polynomial time reduced to L.
- $\text{PSpaceH} := \{L \mid L \text{ is PSPACE-hard}\}$.
- L is called PSPACE-complete if $L \in \text{PSPACE}$ and L is PSPACE-hard.
- $\text{PSpaceC} := \text{PSPACE} \cap \text{PSpaceH}$.

Theorem

If $L' \leq_P L$ and $L' \in \text{PSpaceH}$, then $L \in \text{PSpaceH}$.

The proof is the same as for NP-hard.

How to prove that L is PSPACE-complete?

- First prove that $L \in$ PSPACE: give an algorithm that uses polynomial space for each input.
- Then: pick a well-known $L' \in$ PSpaceH and show $L' \leq_P L$.

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 $x = 0$ we can choose $y = 1$ and for $x = 1$ 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.

NB.

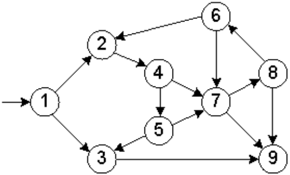
- The “certificate” for QBF(φ) is not just a choice of 0 / 1 for every $\exists$, but a choice depending on the $\forall$ in front of the $\exists$.
- The proof that QBF is PSPACE-hard uses a translation of Turing Machines to QBF.

Some variations on QBF

- Note that SAT $\leq_P$ QBF: given φ add $\exists x$ in front of φ for all atoms x in φ.
- If we limit QBF to prenex fomulas, that have all quantifiers in front, it is still PSPACE-complete.
- If we limit QBF to alternating prenex fomulas, that have alternating $\forall/\exists$ in front, it is still PSPACE-complete.
- If we limit the "body" of the QBF to be a 3CNF, it is still PSPACE-complete.
- A "proof" of $\forall x_1 \exists y_1 \ldots \forall x_n \exists y_n (\varphi)$ amounts to making n choices, which amounts to a "certificate" of size 2^n.
- A formula like $\forall x_1 \exists y_1 \ldots \forall x_n \exists y_n (\varphi)$ 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.