Towards More NP-Complete Problems

Starting with \textbf{Sat}, one can readily show more problems \textbf{P} to be NP-complete, each time performing two steps:

1. Show that \textbf{P} \in \text{NP}
2. Find a known NP-complete problem \textbf{P'} and reduce \textbf{P'} \leq_p \textbf{P}

Thousands of problem have now been shown to be NP-complete. (See Garey and Johnson for an early survey)

In this course:

- \text{Sat} \leq_p \text{Clique}
- \text{Sat} \leq_p \text{Independent Set}
- \text{Sat} \leq_p \text{3-Sat}
- \text{Sat} \leq_p \text{Dir. Hamiltonian Path}
- \text{Sat} \leq_p \text{Subset Sum}
- \text{Sat} \leq_p \text{Knapsack}

NP-Completeness of \textbf{3-Sat}

\textbf{3-Sat}: Satisfiability of formulae in CNF with \( \leq 3 \) literals per clause

**Theorem 8.1:** \textbf{3-Sat} is NP-complete.

**Proof:** Hardness by reduction \textbf{Sat} \leq_p \textbf{3-Sat}:

- Given: \( \varphi \) in CNF
- Construct \( \varphi' \) by replacing clauses \( C_i = (L_1 \lor \cdots \lor L_k) \) with \( k > 3 \) by
  \[ C'_i := (L_1 \lor Y_1) \land (\neg Y_1 \lor L_2 \lor Y_2) \land \cdots \land (\neg Y_{k-1} \lor L_k) \]
- Here, the \( Y_i \) are fresh variables for each clause.
- Claim: \( \varphi \) is satisfiable if \( \varphi' \) is satisfiable.
Example

Let \( \varphi := (X_1 \lor X_2 \lor \neg X_3 \lor X_4) \land (\neg X_4 \lor \neg X_2 \lor X_3 \lor \neg X_1) \)

Then \( \varphi' := (X_1 \lor Y_1) \land \neg Y_1 \lor X_2 \lor Y_2) \land \neg Y_2 \lor \neg X_3 \lor Y_3 \land \neg Y_3 \lor X_4 \land \neg X_4 \lor Z_1 \land \neg Z_1 \lor \neg X_2 \lor Z_2 \land \neg Z_2 \lor X_5 \lor Z_3 \land \neg Z_3 \lor \neg X_1 \)

Proving NP-Completeness of 3-Sat

\(\Rightarrow\) Show that if \( \varphi' \) is satisfiable then \( \varphi \) is satisfiable.

Suppose \( \beta \) is a satisfying assignment for \( \varphi' \).
Then \( \beta \) satisfies \( \varphi \).

Let \( C = (L_1 \lor \cdots \lor L_k) \) be a clause of \( \varphi \).

1. If \( k \leq 3 \) then \( C \) is a clause of \( \varphi' \).
2. If \( k > 3 \) then

   \[
   C' = (L_1 \lor Y_1) \land (\neg Y_1 \lor L_2 \lor Y_2) \land \cdots \land (\neg Y_{k-1} \lor L_k) \in \varphi'
   \]

   \( \beta \) must satisfy at least one \( L_i, 1 \leq i \leq k \).

Case (2) follows since, if \( \beta(L_i) = 0 \) for all \( i \leq k \) then \( C' \) can be reduced to

\[
C' = (Y_1 \land (\neg Y_1 \lor Y_2) \land \cdots \land (\neg Y_{k-1}) \land Y_1 \land (Y_1 \rightarrow Y_2) \land \cdots \land (Y_{k-2} \rightarrow Y_{k-1}) \land \neg Y_{k-1}
\]

which is not satisfiable.

Proving NP-Completeness of Directed Hamiltonian Path

Input: A directed graph \( G \).
Problem: Is there a directed path in \( G \) containing every vertex exactly once?

Theorem 8.2: Directed Hamiltonian Path is NP-complete.

Proof:

1. Directed Hamiltonian Path \( \in \text{NP}; \)
   Take the path to be the certificate.

2. Directed Hamiltonian Path is NP-hard:
   3-Sat \( \leq_p \) Directed Hamiltonian Path
Digression: How to design reductions

Task: Show that problem \( P \) (Directed Hamiltonian Path) is NP-hard.

- Arguably, the most important part is to decide where to start from. That is, which problem to reduce to Directed Hamiltonian Path?

Considerations:
- Is there an NP-complete problem similar to \( P \)? (for example, Clique and Independent Set)
- It is not always beneficial to choose a problem of the same type (for example, reducing a graph problem to a graph problem)
  - For instance, Clique, Independent Set are “local” problems (is there a set of vertices inducing some structure)
  - Hamiltonian Path is a global problem (find a structure – the Hamiltonian path – containing all vertices)

How to design the reduction:
- Does your problem come from an optimisation problem? If so: a maximisation problem? a minimisation problem?
- Learn from examples, have good ideas.

NP-Completeness of Directed Hamiltonian Path

**Proof:** (Proof idea): (see blackboard for details)

Let \( \phi := \bigwedge_{i=1}^{k} C_i \) and \( C_i := (L_{i,1} \lor L_{i,2} \lor L_{i,3}) \)

- For each variable \( X \) occurring in \( \phi \), we construct a directed graph (“gadget”) that allows only two Hamiltonian paths: “true” and “false”
- Gadgets for each variable are “chained” in a directed fashion, so that all variables must be assigned one value
- Clauses are represented by vertices that are connected to the gadgets in such a way that they can only be visited on a Hamiltonian path that corresponds to an assignment where they are true

Details are also given in [Sipser, Theorem 7.46].

**Example 8.3:** \( \phi := C_1 \land C_2 \) where \( C_1 := (X \lor \neg Y \lor Z) \) and \( C_2 := (\neg X \lor Y \lor \neg Z) \)
(see blackboard)

Towards More NP-Complete Problems

Starting with Sat, one can readily show more problems \( P \) to be NP-complete, each time performing two steps:

1. Show that \( P \in \text{NP} \)
2. Find a known NP-complete problem \( P' \) and reduce \( P' \leq_p P \)

Thousands of problems have now been shown to be NP-complete. (See Garey and Johnson for an early survey)

In this course:

\[ \leq_p \text{Clique} \leq_p \text{Independent Set} \leq_p \text{Sat} \leq_p \text{3-Sat} \leq_p \text{Dir. Hamiltonian Path} \leq_p \text{Subset Sum} \leq_p \text{Knapsack} \]
**NP-Completeness of **\textbf{Subset Sum}

\textbf{Subset Sum}

Input: A collection\(^1\) of positive integers \(S = \{a_1, \ldots, a_k\}\) and a target integer \(t\).

Problem: Is there a subset \(T \subseteq S\) such that \(\sum_{i \in T} a_i = t\)?

\textbf{Theorem 8.4:} \textbf{Subset Sum} is NP-complete.

\textbf{Proof:}

(1) \textbf{Subset Sum} \(\in\) NP: Take \(T\) to be the certificate.

(2) \textbf{Subset Sum} is NP-hard: \(\text{Sat} \leq_p \text{Subset Sum}\)

\(^1\) This "collection" is supposed to be a multi-set, i.e., we allow the same number to occur several times. The solution "subset" can likewise use numbers multiple times, but not more often than they occurred in the given collection.

---

**Example**

\[(X_1 \vee X_2 \vee X_3) \wedge (\neg X_1 \vee \neg X_4) \wedge (X_4 \vee X_5 \vee \neg X_2 \vee \neg X_3)\]

<table>
<thead>
<tr>
<th>(X_1)</th>
<th>(X_2)</th>
<th>(X_3)</th>
<th>(X_4)</th>
<th>(X_5)</th>
<th>(C_1)</th>
<th>(C_2)</th>
<th>(C_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\(t = 1 1 1 1 1 3 2 4\)
**NP-Completeness of** \textit{Subset Sum} \\

Let \( \varphi := \bigwedge C_i \) \( i \) \( C_i \): clauses \\

Show: If \( \varphi \) is satisfiable, then there is \( T \subseteq S \) with \( \sum_{s \in T} s = t \).

Let \( \beta \) be a satisfying assignment for \( \varphi \)

Set \( T_1 := \{ \ell_i \mid \beta(X_i) = 1, \ 1 \leq i \leq m \} \) \( T_2 \) := \( \{ \ell_i \mid \beta(X_i) = 0, \ 1 \leq i \leq m \} \)

Further, for each clause \( C_i \) let \( n_i \) be the number of satisfied literals in \( C_i \) (with resp. to \( \beta \)).

Set \( T_2 := \{ m_{i,j} \mid 1 \leq i \leq k, \ 1 \leq j \leq |C_i| - n_i \} \)

and define \( T := T_1 \cup T_2 \).

It follows: \( \sum_{s \in T} s = t \)

**Example** 

\[(X_1 \lor X_2 \lor X_3) \land (\neg X_1 \lor \neg X_4) \land (X_4 \lor X_5 \lor \neg X_2 \lor \neg X_3)\]

<table>
<thead>
<tr>
<th>( T_1 )</th>
<th>( T_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_1 )</td>
<td>( f_1 )</td>
</tr>
<tr>
<td>( f_2 )</td>
<td>( f_2 )</td>
</tr>
<tr>
<td>( f_3 )</td>
<td>( f_3 )</td>
</tr>
<tr>
<td>( f_4 )</td>
<td>( f_4 )</td>
</tr>
<tr>
<td>( f_5 )</td>
<td>( f_5 )</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
m_{1,1} &= 1 \ 0 \\
m_{1,2} &= 1 \ 0 \\
m_{2,1} &= 0 \ 1 \\
m_{3,1} &= 0 \ 1 \\
m_{3,2} &= 0 \ 1 \\
m_{3,3} &= 0 \ 1 \\
t &= 1 \ 1 \ 1 \ 1 \ 1 \ 3 \ 2 \ 4
\end{align*}
\]

**NP-Completeness of** \textit{Subset Sum} \\

Let \( \varphi := \bigwedge C_i \) \( i \) \( C_i \): clauses \\

Show: If there is \( T \subseteq S \) with \( \sum_{s \in T} s = t \), then \( \varphi \) is satisfiable.

Let \( T \subseteq S \) such that \( \sum_{s \in T} s = t \)

Define \( \beta(X_i) = \begin{cases} 1 & \text{if } t_i \in T \\ 0 & \text{if } f_i \in T \end{cases} \)

This is well defined as for all \( i: t_i \in T \) or \( f_i \in T \) but not both.

Further, for each clause, there must be one literal set to 1 as for all \( i \), the \( m_{i,j} \in S \) do not sum up to the number of literals in the clause. \( \square \)
Towards More NP-Complete Problems

Starting with Sat, one can readily show more problems P to be NP-complete, each time performing two steps:

1. Show that P ∈ NP
2. Find a known NP-complete problem P' and reduce P' ≤p P

Thousands of problem have now been shown to be NP-complete. (See Garey and Johnson for an early survey)

In this course:

\[ \text{Subset Sum} \leq_p \text{Knapsack} \]

\[ \text{NP-completeness of Knapsack} \]

**Theorem 8.5**: Knapsack is NP-complete.

**Proof**:

1. Knapsack ∈ NP: Take \( T \) to be the certificate.
2. Knapsack is NP-hard: Subset Sum ≤p Knapsack

A Polynomial Time Algorithm for Knapsack

Knapsack can be solved in time \( O(\ell n) \) using dynamic programming

**Initialisation**:
- Create an \( (\ell + 1) \times (n + 1) \) matrix \( M \)
- Set \( M(w, 0) := 0 \) for all \( w \leq \ell \) and \( M(0, i) := 0 \) for all \( i \)

**Computation**: Assign further \( M(w, i) \) to be the largest total value obtainable by selecting from the first \( i \) items with weight limit \( w \):

\[ M(w, i + 1) := \max \{ M(w, i), M(w - w_{i+1}, i) + v_{i+1} \} \]

Here, if \( w - w_{i+1} < 0 \) we always take \( M(w, i) \).

**Acceptance**: If \( M \) contains an entry \( \geq t \), accept. Otherwise reject.
Example
Input $I = \{1, 2, 3, 4\}$ with
|
| Values: | $v_1 = 1$ | $v_2 = 3$ | $v_3 = 4$ | $v_4 = 2$ |
| Weight: | $w_1 = 1$ | $w_2 = 1$ | $w_3 = 3$ | $w_4 = 2$ |
| Weight limit: $\ell = 5$ | Target value: $t = 7$ |

<table>
<thead>
<tr>
<th>weight limit $w$</th>
<th>$i = 0$</th>
<th>$i = 1$</th>
<th>$i = 2$</th>
<th>$i = 3$</th>
<th>$i = 4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

Set $M(w, 0) := 0$ for all $1 \leq w \leq \ell$ and $M(0, i) := 0$ for all $1 \leq i \leq n$ For $i = 0, 1, \ldots, n - 1$ set $M(w, i + 1) := \max \{M(w, i), M(w - w_{i+1}, i) + v_{i+1}\}$

A Polynomial Time Algorithm for Knapsack

Knapsack can be solved in time $O(n\ell)$ using dynamic programming

Initialisation:
- Create an $(\ell + 1) \times (n + 1)$ matrix $M$
- Set $M(w, 0) := 0$ for all $1 \leq w \leq \ell$ and $M(0, i) := 0$ for all $1 \leq i \leq n$

Computation: Assign further $M(w, i)$ to be the largest total value obtainable by selecting from the first $i$ items with weight limit $w$:
For $i = 0, 1, \ldots, n - 1$ set $M(w, i + 1)$ as

$$M(w, i + 1) := \max \{M(w, i), M(w - w_{i+1}, i) + v_{i+1}\}$$

Here, if $w - w_{i+1} < 0$ we always take $M(w, i)$.

Acceptance: If $M$ contains an entry $\geq t$, accept. Otherwise reject.

Did we prove $P = NP$?

Summary:
- Theorem 5: Knapsack is NP-complete
- Knapsack can be solved in time $O(n\ell)$ using dynamic programming

What went wrong?

Knapsack

- Input: A set $I := \{1, \ldots, n\}$ of items
- each of value $v_i$ and weight $w_i$ for $1 \leq i \leq n$,
- target value $t$ and weight limit $\ell$

Problem: Is there $T \subseteq I$ such that
$$\sum_{i \in T} v_i \geq t \text{ and } \sum_{i \in T} w_i \leq \ell?$$
Pseudo-Polynomial Time

The previous algorithm is not sufficient to show that Knapsack is in P

- The algorithm fills a \((\ell + 1) \times (n + 1)\) matrix \(M\)
- The size of the input to Knapsack is \(O(n \log \ell)\)

\(\leadsto\) the size of \(M\) is not bounded by a polynomial in the length of the input!

Definition 8.6 (Pseudo-Polynomial Time): Problems decidable in time polynomial in the sum of the input length and the value of numbers occurring in the input.

Equivalently: Problems decidable in polynomial time when using unary encoding for all numbers in the input.

- If Knapsack is restricted to instances with \(\ell \leq p(n)\) for a polynomial \(p\), then we obtain a problem in P.
- Knapsack is in polynomial time for unary encoding of numbers.

Strong NP-completeness

Pseudo-Polynomial Time: Algorithms polynomial in the maximum of the input length and the value of numbers occurring in the input.

Examples:
- Knapsack
- Subset Sum

Strong NP-completeness: Problems which remain NP-complete even if all numbers are bounded by a polynomial in the input length (equivalently: even for unary coding of numbers).

Examples:
- Clique
- SAT
- Hamiltonian Cycle
- ...

Note: Showing SAT \(\leq_p\) Subset Sum required exponentially large numbers.

The Class coNP

Recall that coNP is the complement class of NP.

Definition 8.7:
- For a language \(L \subseteq \Sigma^*\) let \(\overline{L} := \Sigma^* \setminus L\) be its complement
- For a complexity class \(C\), we define \(\text{co}C := \{L \mid \overline{L} \in C\}\)
- In particular coNP := \(\{L \mid \overline{L} \in NP\}\)

A problem belongs to coNP, if no-instances have short certificates.

Examples:
- No Hamiltonian Path: Does the graph \(G\) not have a Hamiltonian path?
- Tautology: Is the propositional logic formula \(\varphi\) a tautology (true under all assignments)?
- ...
Definition 8.8: A language \( C \in \text{coNP} \) is coNP-complete, if \( L \leq_p C \) for all \( L \in \text{coNP} \).

Theorem 8.9:
(1) \( P = \text{coP} \)
(2) Hence, \( P \subseteq \text{NP} \cap \text{coNP} \)

Open questions:
- \( \text{NP} = \text{coNP} \)?
  Most people do not think so.

- \( P = \text{NP} \cap \text{coNP} \)?
  Again, most people do not think so.

Example: Chess Problems
Mate in 3 moves; White's turn

Mate in 262 moves; White's turn

Summary and Outlook
3-Sat and Hamiltonian Path are also NP-complete

So are Subset Sum and Knapsack, but only if numbers are encoded efficiently (pseudo-polynomial time)

There do not seem to be polynomial certificates for coNP instances; and for some problems there seem to be certificates neither for instances nor for non-instances

What's next?
- Space
- Games
- Relating complexity classes