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:

\[
\begin{align*}
\text{Sat} & \leq_p \text{Clique} & \leq_p \text{Independent Set} \\
\text{Sat} & \leq_p \text{3-Sat} & \leq_p \text{Dir. Hamiltonian Path} \\
\leq_p \text{Subset Sum} & \leq_p \text{Knapsack}
\end{align*}
\]
3-Sat, Hamiltonian Path, and Subset Sum
NP-Completeness of \(3\text{-SAT}\)

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

**Theorem 8.1:** \(3\text{-SAT}\) is NP-complete.

**Proof:** Hardness by reduction \(\text{SAT} \leq_p 3\text{-SAT}\):

- 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 \(Y_j\) are fresh variables for each clause.
- **Claim:** \(\varphi\) is satisfiable iff \(\varphi'\) is satisfiable.
Example

Let $\varphi := (X_1 \lor X_2 \lor \neg X_3 \lor X_4) \land (\neg X_4 \lor \neg X_2 \lor X_5 \lor \neg X_1)$

Then $\varphi' := (X_1 \lor Y_1) \land$

$(\neg Y_1 \lor X_2 \lor Y_2) \land$

$(\neg Y_2 \lor X_3 \lor Y_3) \land$

$(\neg Y_3 \lor X_4) \land$

$(\neg X_4 \lor Z_1) \land$

$(\neg Z_1 \lor \neg X_2 \lor Z_2) \land$

$(\neg Z_2 \lor X_5 \lor Z_3) \land$

$(\neg Z_3 \lor \neg X_1)$
“⇒” Given $\varphi := \bigwedge_{i=1}^{m} C_i$ with clauses $C_i$, show that if $\varphi$ is satisfiable then $\varphi'$ is satisfiable.

For a satisfying assignment $\beta$ for $\varphi$, define an assignment $\beta'$ for $\varphi'$:

For each $C := (L_1 \lor \cdots \lor L_k)$, with $k > 3$, in $\varphi$ there is

$$C' = (L_1 \lor Y_1) \land (\neg Y_1 \lor L_2 \lor Y_2) \land \ldots \land (\neg Y_{k-1} \lor L_k) \text{ in } \varphi'$$

As $\beta$ satisfies $\varphi$, there is $i \leq k$ s.th. $\beta(L_i) = 1$ i.e.

$$\begin{align*}
\beta(X) &= 1 \text{ if } L_i = X \\
\beta(X) &= 0 \text{ if } L_i = \neg X
\end{align*}$$

$$\beta'(Y_j) = 1 \quad \text{for } j < i$$

Set $$\beta'(Y_j) = 0 \quad \text{for } j \geq i$$

$$\beta'(X) = \beta(X) \quad \text{for all variables in } \varphi$$

This is a satisfying assignment for $\varphi'$.
"⇐" Show that if $\varphi'$ is satisfiable then so is $\varphi$

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

$$C' = (Y_1) \land (\neg Y_1 \lor Y_2) \land \cdots \land (\neg Y_{k-1})$$

$$\equiv Y_1 \land (Y_1 \rightarrow Y_2) \land \cdots \land (Y_{k-2} \rightarrow Y_{k-1}) \land \neg Y_{k-1}$$

which is not satisfiable.

$\square$
NP-Completeness of Directed Hamiltonian Path

**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.
NP-Completeness of **Directed Hamiltonian Path**

**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** ∈ NP:
   Take the path to be the certificate.
Digression: How to design reductions

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

- Arguably, the most important part is to decide where to start from.
  That is, which problem to reduce to \text{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, \textsc{Clique} and \textsc{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, \textsc{Clique}, \textsc{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)
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**

**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 $\in$ NP:**
   - Take the path to be the certificate.
NP-Completeness of **Directed Hamiltonian Path**

**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** $\in$ NP:
   - Take the path to be the certificate.

2. **Directed Hamiltonian Path** is NP-hard:
   - $\text{3-SAT} \leq_p \text{Directed Hamiltonian Path}$
NP-Completeness of Directed Hamiltonian Path

Proof (Proof idea): (see blackboard for details)
Let \( \varphi := \bigwedge_{i=1}^{k} C_i \) and \( C_i := (L_{i,1} \lor L_{i,2} \lor L_{i,3}) \)

- For each variable \( X \) occurring in \( \varphi \), 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: \( \varphi := C_1 \land C_2 \) 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 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:

\begin{align*}
&\leq_p \text{Clique} & \leq_p \text{Independent Set} \\
&\text{Sat} \leq_p 3\text{-Sat} & \leq_p \text{Dir. Hamiltonian Path} \\
&\leq_p \text{Subset Sum} & \leq_p \text{Knapsack}
\end{align*}
NP-Completeness of **Subset Sum**

**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 \( \sum_{a_i \in T} a_i = t \)?

---

**Theorem 8.4:** **Subset Sum** is NP-complete.

**Proof:**

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

2. **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

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

<table>
<thead>
<tr>
<th></th>
<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>(t_1)</td>
<td>1 0 0 0 0</td>
<td>1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(f_1)</td>
<td>1 0 0 0 0</td>
<td>0 1 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(t_2)</td>
<td>1 0 0 0 0</td>
<td>1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(f_2)</td>
<td>1 0 0 0 0</td>
<td>0 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(t_3)</td>
<td>1 0 0 1 0</td>
<td>0 1 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(f_3)</td>
<td>1 0 0 0 0</td>
<td>1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(t_4)</td>
<td>1 0 0 0 1</td>
<td>0 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(f_4)</td>
<td>1 0 0 1 0</td>
<td>0 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(t_5)</td>
<td>1 0 0 1</td>
<td>0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(f_5)</td>
<td>1 0 0 0</td>
<td>0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[m_{1,1} = 1 0 0\]
\[m_{1,2} = 1 0 0\]
\[m_{2,1} = 0 1 0\]
\[m_{3,1} = 0 0 1\]
\[m_{3,2} = 0 0 1\]
\[m_{3,3} = 0 0 1\]

<table>
<thead>
<tr>
<th></th>
<th>(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 1 1 1</td>
</tr>
<tr>
<td>3 2 4</td>
<td></td>
</tr>
</tbody>
</table>
**SAT \leq_p SUBSET SUM**

**Given:** \( \varphi := C_1 \land \cdots \land C_k \) in conjunctive normal form.

(w.l.o.g. at most 9 literals per clause)

Let \( X_1, \ldots, X_n \) be the variables in \( \varphi \). For each \( X_i \) let

\[
\begin{align*}
t_i &:= a_1 \cdots a_n c_1 \cdots c_k \quad \text{where } a_j := \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \quad \text{and } c_j := \begin{cases} 1 & X_i \text{ occurs in } C_j \\ 0 & \text{otherwise} \end{cases} \\

f_i &:= a_1 \cdots a_n c_1 \cdots c_k \quad \text{where } a_j := \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \quad \text{and } c_j := \begin{cases} 1 & \neg X_i \text{ occurs in } C_j \\ 0 & \text{otherwise} \end{cases}
\end{align*}
\]
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>(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>(t_1)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(f_1)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(t_2)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(f_2)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(t_3)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(f_3)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(t_4)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(f_4)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(t_5)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(f_5)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\(| m_{1,1} | = | 1 0 0 |
| m_{1,2} | = | 1 0 0 |
| m_{2,1} | = | 0 1 0 |
| m_{3,1} | = | 0 0 1 |
| m_{3,2} | = | 0 0 1 |
| m_{3,3} | = | 0 0 1 |

\(| t | = | 1 1 1 1 1 3 2 4 |
**SAT \leq_p SUBSET SUM**

Further, for each clause $C_i$ take $r := |C_i| - 1$ integers $m_{i,1}, \ldots, m_{i,r}$

where $m_{i,j} := c_i \ldots c_k$ with $c_\ell := \begin{cases} 1 & \ell = i \\ 0 & \ell \not= i \end{cases}$

**Definition of S:** Let

$$S := \{t_i, f_i \mid 1 \leq i \leq n\} \cup \{m_{i,j} \mid 1 \leq i \leq k, \quad 1 \leq j \leq |C_i| - 1\}$$

**Target:** Finally, choose as target

$$t := a_1 \ldots a_n c_1 \ldots c_k \text{ where } a_i := 1 \text{ and } c_i := |C_i|$$

**Claim:** There is $T \subseteq S$ with $\sum_{a_i \in T} a_i = t$ iff $\varphi$ is satisfiable.
Example

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

\[
\begin{array}{cccccc}
X_1 & X_2 & X_3 & X_4 & X_5 & C_1 & C_2 & C_3 \\
\hline
\end{array}
\]

\[
\begin{array}{cccc}
t_1 &=& 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
f_1 &=& 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
t_2 &=& 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
f_2 &=& 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
t_3 &=& 1 & 0 & 0 & 1 & 0 & 0 \\
f_3 &=& 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
t_4 &=& 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
f_4 &=& 1 & 0 & 0 & 1 & 0 & 0 \\
t_5 &=& 1 & 0 & 0 & 1 & 0 & 0 \\
f_5 &=& 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\hline
m_{1,1} &=& 1 & 0 & 0 \\
m_{1,2} &=& 1 & 0 & 0 \\
m_{2,1} &=& 0 & 1 & 0 \\
m_{3,1} &=& 0 & 0 & 1 \\
m_{3,2} &=& 0 & 0 & 1 \\
m_{3,3} &=& 0 & 0 & 1 \\
\hline
t &=& 1 & 1 & 1 & 1 & 3 & 2 & 4
\end{array}
\]
NP-Completeness of \textbf{Subset Sum}

Let $\varphi := \bigwedge C_i$ \hspace{1cm} $C_i$: 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 $T_1 := \{t_i \mid \beta(X_i) = 1, \ 1 \leq i \leq m\} \cup \{f_i \mid \beta(X_i) = 0, \ 1 \leq i \leq m\}$

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

Set $T_2 := \{m_{i,j} \mid 1 \leq i \leq k, \ 1 \leq j \leq |C_i| - r_i\}$

and define $T := T_1 \cup T_2$.

It follows: $\sum_{s \in T} s = t$
NP-Completeness of \textbf{Subset Sum}

\textbf{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. \hfill \square
Towards More NP-Complete Problems

Starting with $\text{SAT}$, 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 problem 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 \textsc{Knapsack}

\textbf{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

\[ \sum_{i \in T} v_i \geq t \text{ and } \sum_{i \in T} w_i \leq \ell \]

\textbf{Theorem 8.5:} \textsc{Knapsack} is NP-complete.
NP-completeness of **Knapsack**

**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 $\sum_{i \in T} v_i \geq t$ and $\sum_{i \in T} w_i \leq \ell$?

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

**Proof:**

1. **Knapsack** $\in$ NP: Take $T$ to be the certificate.
2. **Knapsack** is NP-hard: **Subset Sum** $\leq_p$ **Knapsack**
SUBSET SUM $\leq_p$ KNAPSACK

Given: $S := \{a_1, \ldots, a_n\}$ collection of positive integers

Subset Sum: $t$ target integer

Problem: Is there a subset $T \subseteq S$ such that $\sum_{a_i \in T} a_i = t$?
Subset Sum $\leq_p$ Knapsack

Given: $S := \{a_1, \ldots, a_n\}$ collection of positive integers

Subset Sum: $t$ target integer

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

Reduction: From this input to Subset Sum construct

- set of items $I := \{1, \ldots, n\}$
- weights and values $v_i = w_i = a_i$ for all $1 \leq i \leq n$
- target value $t' := t$ and weight limit $\ell := t$
**Subset Sum \( \leq_p \) Knapsack**

Given: \( S := \{a_1, \ldots, a_n\} \) collection of positive integers

Subset Sum: \( t \) target integer

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

Reduction: From this input to **Subset Sum** construct

- set of items \( I := \{1, \ldots, n\} \)
- weights and values \( v_i = w_i = a_i \) for all \( 1 \leq i \leq n \)
- target value \( t' := t \) and weight limit \( \ell := t \)

Clearly: For every \( T \subseteq S \)

\[
\sum_{a_i \in T} a_i = t \quad \text{iff} \quad \sum_{a_i \in T} v_i \geq t' = t
\]

\[
\sum_{a_i \in T} w_i \leq \ell = t
\]

Hence: The reduction is correct and in polynomial time.
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 $1 \leq i \leq n$
Example

Input \( I = \{1, 2, 3, 4\} \) with

Values: \( v_1 = 1 \quad v_2 = 3 \quad v_3 = 4 \quad v_4 = 2 \)

Weight: \( w_1 = 1 \quad w_2 = 1 \quad w_3 = 3 \quad w_4 = 2 \)

Weight limit: \( \ell = 5 \) \quad Target value: \( t = 7 \)

<table>
<thead>
<tr>
<th>weight limit ( w )</th>
<th>max. total value from first ( i ) items</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i = 0 )</td>
<td>( i = 1 )</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Set \( M(w, 0) := 0 \) for all \( 1 \leq w \leq \ell \) and \( M(0, i) := 0 \) for all \( 1 \leq i \leq n \).
Example

Input $I = \{1, 2, 3, 4\}$ with

Values: $v_1 = 1 \quad v_2 = 3 \quad v_3 = 4 \quad v_4 = 2$

Weight: $w_1 = 1 \quad w_2 = 1 \quad w_3 = 3 \quad w_4 = 2$

Weight limit: $\ell = 5$ \quad Target value: $t = 7$

<table>
<thead>
<tr>
<th>weight limit $w$</th>
<th>max. total value from first $i$ items</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

Set $M(w, 0) := 0$ for all $1 \leq w \leq \ell$ and $M(0, i) := 0$ for all $1 \leq i \leq n$
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 $1 \leq i \leq n$
A Polynomial Time Algorithm for \textbf{KNAPSACK}

\textbf{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.
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>max. total value from first $i$ items</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$i = 0$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

For $i = 0, 1, \ldots, n - 1$ set $M(w, i + 1) := \max\{M(w, i), M(w - w_{i+1}, i) + v_{i+1}\}$
Example

Input $I = \{1, 2, 3, 4\}$ with

Values: $v_1 = 1 \quad v_2 = 3 \quad v_3 = 4 \quad v_4 = 2$

Weight: $w_1 = 1 \quad w_2 = 1 \quad w_3 = 3 \quad w_4 = 2$

Weight limit: $\ell = 5$ \quad Target value: $t = 7$

<table>
<thead>
<tr>
<th>weight limit $w$</th>
<th>max. total value from first $i$ items</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i = 0$</td>
<td>$i = 1$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

For $i = 0, 1, \ldots, n - 1$ set $M(w, i + 1) := \max\{M(w, i), \ M(w - w_{i+1}, i) + v_{i+1}\}$
**Example**

Input $I = \{1, 2, 3, 4\}$ with

**Values:** $v_1 = 1 \quad v_2 = 3 \quad v_3 = 4 \quad v_4 = 2$

**Weight:** $w_1 = 1 \quad w_2 = 1 \quad w_3 = 3 \quad w_4 = 2$

**Weight limit:** $\ell = 5 \quad$ **Target value:** $t = 7$

<table>
<thead>
<tr>
<th>weight limit $w$</th>
<th>max. total value from first $i$ items</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i = 0$</td>
<td>$i = 1$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

For $i = 0, 1, \ldots, n - 1$ set $M(w, i + 1) := \max \{M(w, i), M(w - w_{i+1}, i) + v_{i+1}\}$
Example

Input $I = \{1, 2, 3, 4\}$ with

Values: $v_1 = 1 \quad v_2 = 3 \quad v_3 = 4 \quad v_4 = 2$

Weight: $w_1 = 1 \quad w_2 = 1 \quad w_3 = 3 \quad w_4 = 2$

Weight limit: $\ell = 5$ \quad Target value: $t = 7$

<table>
<thead>
<tr>
<th>weight limit $w$</th>
<th>max. total value from first $i$ items</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i = 0$</td>
<td>$i = 1$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

For $i = 0, 1, \ldots, n - 1$ set $M(w, i + 1) := \max \{M(w, i), M(w - w_{i+1}, i) + v_{i+1}\}$
Input \( I = \{1, 2, 3, 4\} \) with

Values: \( v_1 = 1 \quad v_2 = 3 \quad v_3 = 4 \quad v_4 = 2 \)

Weight: \( w_1 = 1 \quad w_2 = 1 \quad w_3 = 3 \quad w_4 = 2 \)

Weight limit: \( \ell = 5 \quad \) Target value: \( t = 7 \)

<table>
<thead>
<tr>
<th>weight limit ( w )</th>
<th>max. total value from first ( i ) items</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( i = 0 )</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

For \( i = 0, 1, \ldots, n - 1 \) set \( M(w, i + 1) := \max \{M(w, i), M(w - w_{i+1}, i) + v_{i+1}\} \)
Example

Input $I = \{1, 2, 3, 4\}$ with

Values: $v_1 = 1 \quad v_2 = 3 \quad v_3 = 4 \quad v_4 = 2$

Weight: $w_1 = 1 \quad w_2 = 1 \quad w_3 = 3 \quad w_4 = 2$

Weight limit: $\ell = 5$  Target value: $t = 7$

<table>
<thead>
<tr>
<th>weight limit $w$</th>
<th>max. total value from first $i$ items</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i = 0$</td>
<td>$i = 1$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

For $i = 0, 1, \ldots, n - 1$ set $M(w, i + 1) := \max \{M(w, i), M(w - w_{i+1}, i) + v_{i+1}\}$
Example

Input $I = \{1, 2, 3, 4\}$ with

Values: $v_1 = 1 \quad v_2 = 3 \quad v_3 = 4 \quad v_4 = 2$

Weight: $w_1 = 1 \quad w_2 = 1 \quad w_3 = 3 \quad w_4 = 2$

Weight limit: $\ell = 5$ \quad 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>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For $i = 0, 1, \ldots, n-1$ set $M(w, i + 1) := \max \{M(w, i), M(w - w_{i+1}, i) + v_{i+1}\}$
Example

Input $I = \{1, 2, 3, 4\}$ with

Values: $v_1 = 1 \quad v_2 = 3 \quad v_3 = 4 \quad v_4 = 2$

Weight: $w_1 = 1 \quad w_2 = 1 \quad w_3 = 3 \quad w_4 = 2$

Weight limit: $\ell = 5$ \hspace{1cm} Target value: $t = 7$

<table>
<thead>
<tr>
<th>weight limit $w$</th>
<th>max. total value from first $i$ items</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i = 0$</td>
<td>$i = 1$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

For $i = 0, 1, \ldots, n-1$ set $M(w, i+1) := \max \{M(w, i), M(w - w_{i+1}, i) + v_{i+1}\}$
Did we prove $P = NP$?

Summary:

- Theorem 8.5: **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
- $\sum_{i \in T} v_i \geq t$ 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 \(O(n \log \ell)\).

\[ \leadsto \text{the size of } M \text{ is not bounded by a polynomial in the length of the input!} \]
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 \(O(n \log \ell)\)

\(\sim\) 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.
Beyond NP
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 \( \text{coNP} = \{ L \mid \overline{L} \in \text{NP} \} \).

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:

- NP = coNP?
  
  Most people do not think so.

- P = NP $\cap$ coNP?
  
  Again, most people do not think so.
Mate in 3 moves; White’s turn
Example: Chess Problems

Mate in 262 moves; White’s turn
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