

# COMPLEXITY THEORY

## Lecture 8: NP-Complete Problems

Markus Krötzsch, Stephan Mennicke, Lukas Gerlach  
Knowledge-Based Systems

TU Dresden, 9th Nov 2023

More recent versions of this slide deck might be available.  
For the most current version of this course, see  
[https://iccl.inf.tu-dresden.de/web/Complexity\\_Theory/en](https://iccl.inf.tu-dresden.de/web/Complexity_Theory/en)

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

In this course:

$$\begin{array}{ll} \leq_p \text{ CLIQUE} & \leq_p \text{ INDEPENDENT SET} \\ \text{SAT} \leq_p \text{ 3-SAT} & \leq_p \text{ DIR. HAMILTONIAN PATH} \\ \leq_p \text{ SUBSET SUM} & \leq_p \text{ KNAPSACK} \end{array}$$

# 3-Sat, Hamiltonian Path, and Subset Sum

# NP-Completeness of **3-SAT**

**3-SAT**: Satisfiability of formulae in CNF with  $\leq 3$  literals per clause

**Theorem 8.1: 3-SAT** is NP-complete.

**Proof:** Hardness by reduction **SAT**  $\leq_p$  **3-SAT**:

- Given:  $\varphi$  in CNF
- Construct  $\varphi'$  by replacing clauses  $C_i = (L_1 \vee \dots \vee L_k)$  with  $k > 3$  by

$$C'_i := (L_1 \vee Y_1) \wedge (\neg Y_1 \vee L_2 \vee Y_2) \wedge \dots \wedge (\neg Y_{k-1} \vee L_k)$$

Here, the  $Y_j$  are fresh variables for each clause.

- **Claim:**  $\varphi$  is satisfiable iff  $\varphi'$  is satisfiable.

# Example

Let  $\varphi := (X_1 \vee X_2 \vee \neg X_3 \vee X_4) \wedge (\neg X_4 \vee \neg X_2 \vee X_5 \vee \neg X_1)$

Then  $\varphi' := (X_1 \vee Y_1) \wedge$

$(\neg Y_1 \vee X_2 \vee Y_2) \wedge$

$(\neg Y_2 \vee \neg X_3 \vee Y_3) \wedge$

$(\neg Y_3 \vee X_4) \wedge$

$(\neg X_4 \vee Z_1) \wedge$

$(\neg Z_1 \vee \neg X_2 \vee Z_2) \wedge$

$(\neg Z_2 \vee X_5 \vee Z_3) \wedge$

$(\neg Z_3 \vee \neg X_1)$

# Proving NP-Completeness of **3-SAT**

“ $\Rightarrow$ ” 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  $C := (L_1 \vee \dots \vee L_k)$ , with  $k > 3$ , in  $\varphi$  there is

$$C' = (L_1 \vee Y_1) \wedge (\neg Y_1 \vee L_2 \vee Y_2) \wedge \dots \wedge (\neg Y_{k-1} \vee L_k) \text{ in } \varphi'$$

As  $\beta$  satisfies  $\varphi$ , there is  $i \leq k$  s.th.  $\beta(L_i) = 1$  i.e.  $\beta(X) = 1$  if  $L_i = X$   
 $\beta(X) = 0$  if  $L_i = \neg X$

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

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  $\varphi'$

# Proving NP-Completeness of 3-SAT

“ $\Leftarrow$ ” Show that if  $\varphi'$  is satisfiable then so is  $\varphi$

Suppose  $\beta$  is a satisfying assignment for  $\varphi'$  – then  $\beta$  satisfies  $\varphi$ :

Let  $C := (L_1 \vee \dots \vee L_k)$  be a clause of  $\varphi$

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

(2) If  $k > 3$  then

$$C' = (L_1 \vee Y_1) \wedge (\neg Y_1 \vee L_2 \vee Y_2) \wedge \dots \wedge (\neg Y_{k-1} \vee L_k) \text{ in } \varphi'$$

$\beta$  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

$$\begin{aligned} C' &= (Y_1) \wedge (\neg Y_1 \vee Y_2) \wedge \dots \wedge (\neg Y_{k-1}) \\ &\equiv Y_1 \wedge (Y_1 \rightarrow Y_2) \wedge \dots \wedge (Y_{k-2} \rightarrow Y_{k-1}) \wedge \neg Y_{k-1} \end{aligned}$$

which is not satisfiable. □

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

## 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**?

# 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)

# 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:

**3-SAT**  $\leq_p$  **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  $C_i := (L_{i,1} \vee L_{i,2} \vee L_{i,3})$

- 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:**  $\varphi := C_1 \wedge C_2$  where  $C_1 := (X \vee \neg Y \vee Z)$  and  $C_2 := (\neg X \vee Y \vee \neg Z)$   
(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 problem have now been shown to be NP-complete.  
(See Garey and Johnson for an early survey)

In this course:

$$\begin{array}{ll} \leq_p \text{ CLIQUE} & \leq_p \text{ INDEPENDENT SET} \\ \text{SAT} \leq_p \text{ 3-SAT} & \leq_p \text{ DIR. HAMILTONIAN PATH} \\ \leq_p \text{ SUBSET SUM} & \leq_p \text{ KNAPSACK} \end{array}$$



# NP-Completeness of **SUBSET SUM**

## **SUBSET SUM**

Input: A collection<sup>1</sup> of positive integers

$S = \{a_1, \dots, a_k\}$  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**  $\in$  NP: Take  $T$  to be the certificate.
- (2) **SUBSET SUM** is NP-hard: **SAT**  $\leq_p$  **SUBSET SUM**

<sup>1</sup>) 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

$$(X_1 \vee X_2 \vee X_3) \wedge (\neg X_1 \vee \neg X_4) \wedge (X_4 \vee X_5 \vee \neg X_2 \vee \neg X_3)$$

		$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$C_1$	$C_2$	$C_3$
$t_1$	=	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
$t_3$	=			1	0	0	1	0	0
$f_3$	=			1	0	0	0	0	1
$t_4$	=				1	0	0	0	1
$f_4$	=				1	0	0	1	0
$t_5$	=					1	0	0	1
$f_5$	=					1	0	0	0
$m_{1,1}$	=						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

**Given:**  $\varphi := C_1 \wedge \dots \wedge C_k$  in conjunctive normal form.

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

Let  $X_1, \dots, X_n$  be the variables in  $\varphi$ . For each  $X_i$  let

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

$$f_i := a_1 \dots a_n c_1 \dots c_k \text{ where } a_j := \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \text{ and } c_j := \begin{cases} 1 & \neg X_i \text{ occurs in } C_j \\ 0 & \text{otherwise} \end{cases}$$

# Example

$$(X_1 \vee X_2 \vee X_3) \wedge (\neg X_1 \vee \neg X_4) \wedge (X_4 \vee X_5 \vee \neg X_2 \vee \neg X_3)$$

		$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$C_1$	$C_2$	$C_3$
$t_1$	=	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
$t_3$	=			1	0	0	1	0	0
$f_3$	=			1	0	0	0	0	1
$t_4$	=				1	0	0	0	1
$f_4$	=				1	0	0	1	0
$t_5$	=					1	0	0	1
$f_5$	=					1	0	0	0
$m_{1,1}$	=						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  $m_{i,1}, \dots, m_{i,r}$

where  $m_{i,j} := c_i \dots c_k$  with  $c_\ell := \begin{cases} 1 & \ell = i \\ 0 & \ell \neq i \end{cases}$

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

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

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

# Example

$$(X_1 \vee X_2 \vee X_3) \wedge (\neg X_1 \vee \neg X_4) \wedge (X_4 \vee X_5 \vee \neg X_2 \vee \neg X_3)$$

		$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$C_1$	$C_2$	$C_3$
$t_1$	=	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
$t_3$	=			1	0	0	1	0	0
$f_3$	=			1	0	0	0	0	1
$t_4$	=				1	0	0	0	1
$f_4$	=				1	0	0	1	0
$t_5$	=					1	0	0	1
$f_5$	=					1	0	0	0
$m_{1,1}$	=						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

# NP-Completeness of **SUBSET SUM**

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

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 **SUBSET SUM**

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

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

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

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. □



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

In this course:

$$\begin{array}{ll} \leq_p \text{ CLIQUE} & \leq_p \text{ INDEPENDENT SET} \\ \text{SAT} \leq_p \text{ 3-SAT} & \leq_p \text{ DIR. HAMILTONIAN PATH} \\ \leq_p \text{ SUBSET SUM} & \leq_p \text{ KNAPSACK} \end{array}$$

# NP-completeness of **KNAPSACK**

## **KNAPSACK**

Input: A set  $I := \{1, \dots, n\}$  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.

# NP-completeness of **KNAPSACK**

## **KNAPSACK**

Input: A set  $I := \{1, \dots, n\}$  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:  $S := \{a_1, \dots, a_n\}$  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:  $S := \{a_1, \dots, a_n\}$  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  $I := \{1, \dots, n\}$
- 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:  $S := \{a_1, \dots, a_n\}$  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  $I := \{1, \dots, n\}$
- 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  $T \subseteq S$

$$\sum_{a_i \in T} a_i = t \quad \text{iff} \quad \begin{aligned} \sum_{a_i \in T} v_i &\geq t' = t \\ \sum_{a_i \in T} w_i &\leq \ell = t \end{aligned}$$

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:  $v_1 = 1$   $v_2 = 3$   $v_3 = 4$   $v_4 = 2$

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

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

weight limit $w$	max. total value from first $i$ items				
	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0					
1					
2					
3					
4					
5					

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:  $v_1 = 1$   $v_2 = 3$   $v_3 = 4$   $v_4 = 2$

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

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

weight limit $w$	max. total value from first $i$ items				
	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0	0	0	0	0	0
1	0				
2	0				
3	0				
4	0				
5	0				

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 **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  $i = 0, 1, \dots, n - 1$  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:  $t = 7$

weight limit $w$	max. total value from first $i$ items				
	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0	0	0	0	0	0
1	0				
2	0				
3	0				
4	0				
5	0				

For  $i = 0, 1, \dots, n - 1$  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:  $v_1 = 1$   $v_2 = 3$   $v_3 = 4$   $v_4 = 2$

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

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

weight limit $w$	max. total value from first $i$ items				
	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0	0	0	0	0	0
1	0	1			
2	0	1			
3	0	1			
4	0	1			
5	0	1			

For  $i = 0, 1, \dots, n - 1$  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:  $v_1 = 1$   $v_2 = 3$   $v_3 = 4$   $v_4 = 2$

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

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

weight limit $w$	max. total value from first $i$ items				
	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0	0	0	0	0	0
1	0	1	3		
2	0	1			
3	0	1			
4	0	1			
5	0	1			

For  $i = 0, 1, \dots, n - 1$  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:  $v_1 = 1$   $v_2 = 3$   $v_3 = 4$   $v_4 = 2$

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

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

weight limit $w$	max. total value from first $i$ items				
	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0	0	0	0	0	0
1	0	1	3		
2	0	1	4		
3	0	1			
4	0	1			
5	0	1			

For  $i = 0, 1, \dots, n - 1$  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:  $v_1 = 1$   $v_2 = 3$   $v_3 = 4$   $v_4 = 2$

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

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

weight limit $w$	max. total value from first $i$ items				
	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0	0	0	0	0	0
1	0	1	3		
2	0	1	4		
3	0	1	4		
4	0	1			
5	0	1			

For  $i = 0, 1, \dots, n - 1$  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:  $v_1 = 1$   $v_2 = 3$   $v_3 = 4$   $v_4 = 2$

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

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

weight limit $w$	max. total value from first $i$ items				
	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0	0	0	0	0	0
1	0	1	3		
2	0	1	4		
3	0	1	4		
4	0	1	4		
5	0	1			

For  $i = 0, 1, \dots, n - 1$  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:  $v_1 = 1$   $v_2 = 3$   $v_3 = 4$   $v_4 = 2$

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

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

weight limit $w$	max. total value from first $i$ items				
	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0	0	0	0	0	0
1	0	1	3		
2	0	1	4		
3	0	1	4		
4	0	1	4		
5	0	1	4		

For  $i = 0, 1, \dots, n - 1$  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:  $v_1 = 1$   $v_2 = 3$   $v_3 = 4$   $v_4 = 2$

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

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

weight limit $w$	max. total value from first $i$ items				
	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0	0	0	0	0	0
1	0	1	3	3	3
2	0	1	4	4	4
3	0	1	4	4	5
4	0	1	4	7	7
5	0	1	4	8	8

For  $i = 0, 1, \dots, n - 1$  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  $I := \{1, \dots, n\}$  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  $O(n \log \ell)$

↪ the size of  $M$  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  $O(n \log \ell)$

↪ 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  $\bar{L} := \Sigma^* \setminus L$  be its complement
- For a complexity class  $C$ , we define  $\text{co}C := \{L \mid \bar{L} \in C\}$
- In particular  $\text{coNP} = \{L \mid \bar{L} \in \text{NP}\}$

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)?
- ...

# coNP-completeness

**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:

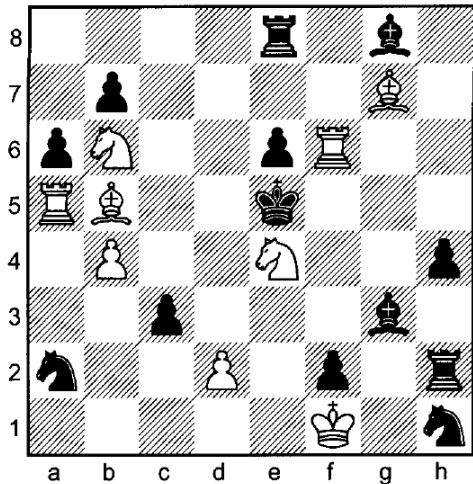
- $\text{NP} = \text{coNP}$ ?

Most people do not think so.

- $P = \text{NP} \cap \text{coNP}$ ?

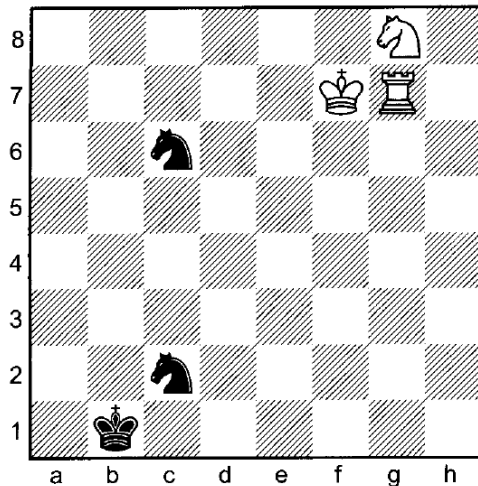
Again, most people do not think so.

# Example: Chess Problems



Mate in 3 moves; White's turn

# Example: Chess Problems



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