NP-完全

計算複雜性理論

Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
布爾諾，捷克

Ondřej Lengál
龍安卓
（龍龖龘）

This material was created with the support of the Czech Ministry of Education, Youth and Sports (project FRVŠ 166/2013/G1).

Outline

- We will talk about the power of nondeterministic computation.
- We will focus on the class of **NP**-complete problems.

Deterministic Computation

Classical algorithms are deterministic, i.e., in every non-final program state, there is exactly one successor state.

- **program state** = program counter + contents of variables
- $\rightsquigarrow$ for every input, there is exactly one run

Nondeterministic Computation

Nondeterministic algorithms

- have access to nondeterministic values

```
1 bool f(unsigned input) {
2   unsigned p = nondet();
3   unsigned q = input / q;    // div
4   if (p * q == input)
5     return true;
6   else
7     return false;
8 }
```

- for one input, there may be more than one run
  - for different return values of `nondet()` calls

Decision Problems

Definition

Let $\mathbb{D}$ be a domain and $\Psi \subseteq \mathbb{D}$. We call Ψ a decision problem (DP) of $\mathbb{D}$.

Let *Alg* be a (deterministic) algorithm. We say that *Alg* decides Ψ iff for all $x \in \mathbb{D}$, *Alg* terminates and returns *true* if $x \in \Psi$ and *false* if $x \notin \Psi$.

Let *Alg* be a **nondeterministic** algorithm. We say that *Alg* decides Ψ iff for all $x \in \mathbb{D}$ it holds that:

- if $x \in \mathbb{D}$, there exists a run of *Alg* that terminates and returns *true*,
- if $x \notin \mathbb{D}$, *Alg* never returns *true*.

Decision Problems

Example

Let

- $\mathbb{N} = \{0, 1, 2, \ldots\}$ be the set of natural numbers and
- $\mathbb{G}$ be the set of all finite graphs $G = (V, E)$ s.t. $V \subseteq \mathbb{N}, E \subseteq V \times V$.

Then

1. $\{(V, E) \mid |V| = 42\}$ is a DP of $\mathbb{G}$,
2. $\{(V, E) \mid (V, E)$ contains a Hamiltonian path$\}$ is a DP of $\mathbb{G}$,
3. $\{((V, E), n) \mid |V| = n\}$ is a DP of $\mathbb{G} \times \mathbb{N}$,
4. $\{((V, E), i, j) \mid \exists$path from i to j in $(V, E)\}$ is a DP of $\mathbb{G} \times \mathbb{N} \times \mathbb{N}$,
5. $\{x \mid x$ is a prime$\}$ is a DP of $\mathbb{N}$, and
6. $\{x \mid x$ is the Gödel number of the proof of $\mathbf{P} \stackrel{?}{=} \mathbf{NP}\}$ is a DP of $\mathbb{N}$.

Time Complexity

Definition

$\mathcal{O}(f(n)) = \{g : \mathbb{N} \to \mathbb{N} \mid \exists c, k \in \mathbb{N}.\ \forall n > k.\ g(n) \leq c \cdot f(n)\}$

Definition

Let $\mathbb{D}$ be a domain. We assume that $\mathbb{D}$ comes with a function $enc : \mathbb{D} \to \{0,1\}^*$ that assigns every element of $x \in \mathbb{D}$ its binary encoding. The size of $x \in \mathbb{D}$, denoted as $|x|$, is defined as $len(enc(x))$.

Let Alg be an algorithm. We define the function $Time_{Alg} : \mathbb{D} \to \mathbb{N}$ s.t. $Time_{Alg}(x)$ is the number of steps taken by the shortest run of Alg deciding x, and $T_{Alg} : \mathbb{N} \to \mathbb{N}$ as $T_{Alg}(n) = \max\{Time_{Alg}(x) \mid |x| = n\}$.

We say that $\mathcal{O}(f(n))$ is the (worst-case) time complexity of Alg if $T_{Alg}(n) \in \mathcal{O}(f(n))$.

Time Complexity

Definition

Let $\mathbb{D}$ be a domain.

- $\text{DTime}[f(n)] = \{\Psi \subseteq \mathbb{D} | \exists \text{det } Alg.\ Alg \text{ decides } \Psi \wedge T_{Alg} \in \mathcal{O}(f(n))\}$
- $\text{NTime}[f(n)] = \{\Psi \subseteq \mathbb{D} | \exists \text{ndet } Alg.\ Alg \text{ decides } \Psi \wedge T_{Alg} \in \mathcal{O}(f(n))\}$

Lemma

$$\text{NTime}[f(n)] \subseteq \text{DTime}[2^{f(n)}]$$

Complexity Classes

Complexity Classes

- classify decision problems according to their inherent complexity

Definition

$$\mathbf{P} = \bigcup_{i \leq 0} \mathrm{DTime}[n^i] \qquad \mathbf{NP} = \bigcup_{i \leq 0} \mathrm{NTime}[n^i]$$

The two classes are sometimes said to denote

- **P** = “practically solvable problems”
- **NP** = “infeasible problems”

Complexity Classes

Complexity Classes

- classify decision problems according to their inherent complexity

Definition

$$\mathbf{P} = \bigcup_{i \leq 0} \mathrm{DTime}[n^i] \qquad \mathbf{NP} = \bigcup_{i \leq 0} \mathrm{NTime}[n^i]$$

The two classes are sometimes said to denote

- **P** = “practically solvable problems”
- **NP** = “infeasible problems”

Open problem:

$$\mathbf{P} \stackrel{?}{=} \mathbf{NP}$$

Completeness

The concept of completeness is one of the most important in complexity theory.

Definition (Hardness, Completeness)

Let **C** be a complexity class. We call a DP Ψ

C-hard if for all $\Phi \in \mathbf{C}$, $\Phi \leq \Psi$,

C-complete if Ψ is **C**-hard and $\Psi \in \mathbf{C}$.

Note: Given DPs P_1 and P_2 of $\mathbb{D}_1$ and $\mathbb{D}_2$ respectively, we use $P_1 \leq P_2$ to denote that there exists a polynomial-time reduction from P_1 to P_2, i.e. that there exists a **PTIME** algorithm computing a function $R : \mathbb{D}_1 \to \mathbb{D}_2$ s.t. $x \in P_1 \iff R(x) \in P_2$.

This means that **C**-complete problems are the hardest problems of **C**.

Motivation

Proving that a DP Ψ is **NP**-complete means that:

- there is probably no fast algorithm for solving Ψ,
- naïve ways for solving Ψ will probably not work,
- heuristics may be necessary for practical algorithms,
- or we may just try to find an approximate solution,
- Richard Karp. *Reducibility Among Combinatorial Problems.* 1972.

SAT

SAT: Is a given propositional formula φ satisfiable?

Theorem (Cook-Levin 1971/73)

SAT *is* **NP***-complete.*

Proof.

- SAT $\in$ **NP** — by constructing an **NPTIME** algorithm deciding SAT.
- SAT is **NP**-hard — by showing that for any **NPTIME** *Alg* and its input x, there is a **PTIME** reduction to a propositional formula φ s.t. φ is satisfiable iff $Alg(x)$ can return *true*. □

CNF

CNF: Is a given propositional formula φ in the conjunctive normal form satisfiable?

Theorem

CNF *is* **NP***-complete.*

Proof.

- CNF is **NP**-hard — from SAT using Tseitin transformation
  - transforms ψ into an equisatisfiable formula φ in CNF,
  - the size of φ grows linearly with the size of ψ,
  - naïve transformation (using De Morgan's laws and distribution) yields exponentially larger formula in the worst case. □

Note: in practice, "SAT" is often used to mean "CNF".

Tseitin Transformation

- φ and ψ are equisatisfiable when φ is satisfiable iff ψ is satisfiable
- naïve transformation using distributivity $\rightsquigarrow$ exponential size
- **Tseytin transformation**: linear size
  1. transform φ to ψ in the negation normal form (NNF) [linear size]
     - NNF: only $\wedge$, $\vee$, and $\neg$ of atoms
  2. transform ψ to a combinatorial circuit
     - **variables** of ψ are inputs
     - every **subformula** corresponds to a logical gate
     - an **auxiliary variable** for every gate (denotes its output)
     - **gates**: small CNF formulae relating inputs (A, B) with the output (C)
     - the result is the conjuction of formulae for all gates [still linear size]
     - one more clause is added denoting that the output value is *true*

Tseitin Transformation

Type	Operation	CNF expression
NOT	$C \equiv \neg A$	$(A \vee C) \wedge (\neg A \vee \neg C)$
AND	$C \equiv A \wedge B$	$(A \vee \neg C) \wedge (B \vee \neg C) \wedge (\neg A \vee \neg B \vee C)$
OR	$C \equiv A \vee B$	$(\neg A \vee C) \wedge (\neg B \vee C) \wedge (A \vee B \vee \neg C)$

Table: Correct behaviour of gates (A, B are inputs, C is output, Z denotes when the gate is operating correctly)

NOT		
A	C	Z
0	0	0
0	1	1
1	0	1
1	1	0

AND			
A	B	C	Z
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

OR			
A	B	C	Z
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

k-CNF (k-SAT)

k-CNF: A restricted version of CNF where each clause has exactly k literals.

Theorem

2CNF $\in$ **P**.

Proof.

Clauses can be rewritten to implications which can be viewed as Horn clauses. There is a **PTIME** algorithm for solving HORNSAT. □

k-CNF (k-SAT)

Theorem

*k-CNF is **NP**-complete for $k \geq 3$.*

Proof.

- 3-CNF is **NP**-hard — by reduction from CNF (similarly for other k). We can transform every clause

$$(a \vee b \vee c \vee \cdots \vee f \vee g)$$

into the conjunction

$$(a \vee b \vee x) \wedge (\neg x \vee c \vee y) \wedge \cdots \wedge (\neg z \vee f \vee g)$$

which is equisatisfiable and only linearly larger. □

3-CNF (3-SAT) is interesting because it is the variant of k-CNF with the lowest k that is **NP**-complete.

CLIQUE

CLIQUE: Given a graph $G = (V, E)$ and $k \in \mathbb{N}$, does G contain a clique (a complete subgraph) of size k?

Theorem

CLIQUE *is* **NP***-complete.*

Proof.

- CLIQUE is **NP**-hard — by reduction from CNF. For a formula $C_1 \wedge \cdots \wedge C_n$ we set $k = n$ and construct an undirected graph $G = (V, E)$ such that

$$V = \{(\sigma, i) \mid \sigma \text{ is a literal and occurs in } C_i\}$$
$$E = \{\{(\sigma, i), (\delta, j)\} \mid i \neq j \wedge \sigma \neq \neg\delta\}$$

□

INDEPENDENT SET

INDEPENDENT SET: Given a graph $B = (W, J)$ and $m \in \mathbb{N}$, does B contain an independent set of vertices (a set of vertices no two of which are adjacent) of size at least m?

Theorem

INDEPENDENT SET *is* **NP***-complete.*

Proof.

- INDEPENDENT SET is **NP**-hard — by reduction from CLIQUE. For a graph $G = (V, E)$ and k, we set $m = k$ and construct

$$B = (V, V^2 \setminus E)$$

□

Note that cliques are independent sets in graphs' complements.

VERTEX COVER

VERTEX COVER: Given a graph $H = (U, F)$ and $l \in \mathbb{N}$, does H have a vertex cover of size at most l? I.e., is there a set of vertices $S \subseteq U$ of size $|S| \leq l$ such that all edges of H are incident with at least one vertex from S?

Theorem

VERTEX COVER *is* **NP***-complete.*

Proof.

- VERTEX COVER is **NP**-hard — by reduction from INDEPENDENT SET. For a graph $B = (W, J)$ and m, we set $l = |W| - m$ and $H = B$. □

Note that a set is independent iff its complement is a vertex cover.

GRAPH COLOURING

GRAPH COLOURING: Given a graph $M = (Y, L)$ and $p \in \mathbb{N}$, can the vertices of M be coloured using p colours such that no two adjacent vertices are assigned the same colour?

Theorem

GRAPH COLOURING $\in$ **P** *for* $p = 2$.

Proof.

- A graph is 2-colourable iff it is bipartite, which can be determined using BFS in linear time. □

GRAPH COLOURING

Theorem

GRAPH COLOURING *is* **NP**-*complete for* $p \geq 3$.

Proof.

- GRAPH COLOURING for $p \geq 3$ is **NP**-hard — by reduction from 3-CNF. For a formula $\varphi_1 \wedge \cdots \wedge \varphi_k$ over variables $x_1, \ldots, x_r$, we set $p = r + 1$ and construct the graph $M = (Y, L)$ in the following way: *Assume the formula*

$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2)$$

GRAPH COLOURING

Theorem

GRAPH COLOURING *is* **NP**-*complete for* $p \geq 3$.

Proof.

- GRAPH COLOURING for $p \geq 3$ is **NP**-hard — by reduction from 3-CNF. For a formula $\varphi_1 \wedge \cdots \wedge \varphi_k$ over variables $x_1, \ldots, x_r$, we set $p = r + 1$ and construct the graph $M = (Y, L)$ in the following way: *Assume the formula*

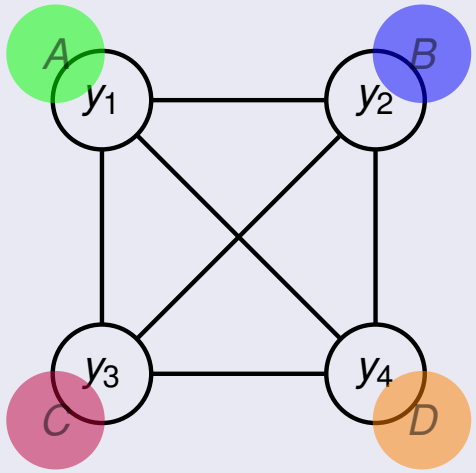
$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2)$$

1. Make sure there are at least 4 variables ($r \geq 4$), otherwise add.
   - *we add* x_4 *to the set of variables* $\rightarrow \{x_1, x_2, x_3, x_4\}$, *and*
   - *set the number of colours* $p = 5$, *call them* $\{A, B, C, D, E\}$.

GRAPH COLOURING

Proof (cont).

2. Create a clique with a node y_i for every variable x_i.

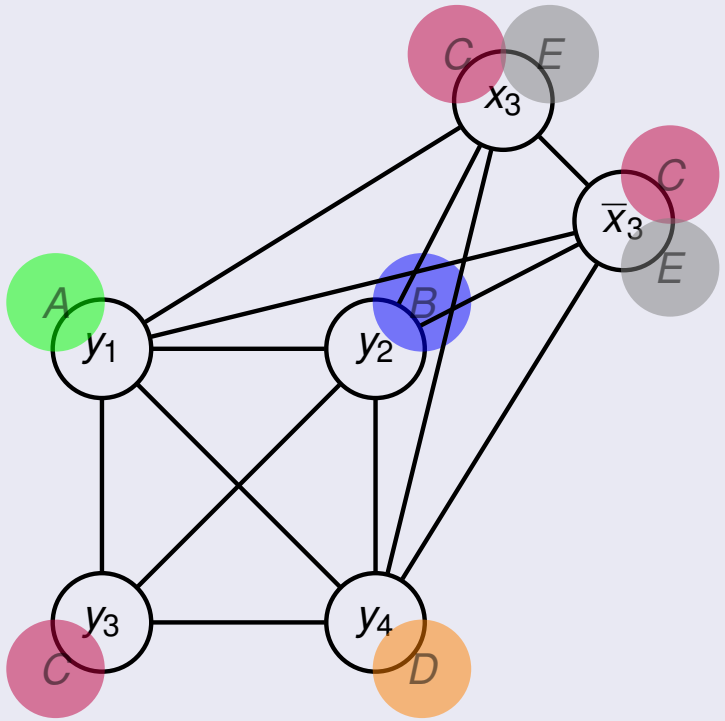


Each node of the clique needs to be coloured with a different colour.

GRAPH COLOURING

Proof (cont).

3. For every variable x_i, add nodes labelled with x_i and $\overline{x_i}$ and connect them with each other and with all y_j, $i \neq j$, from the clique.



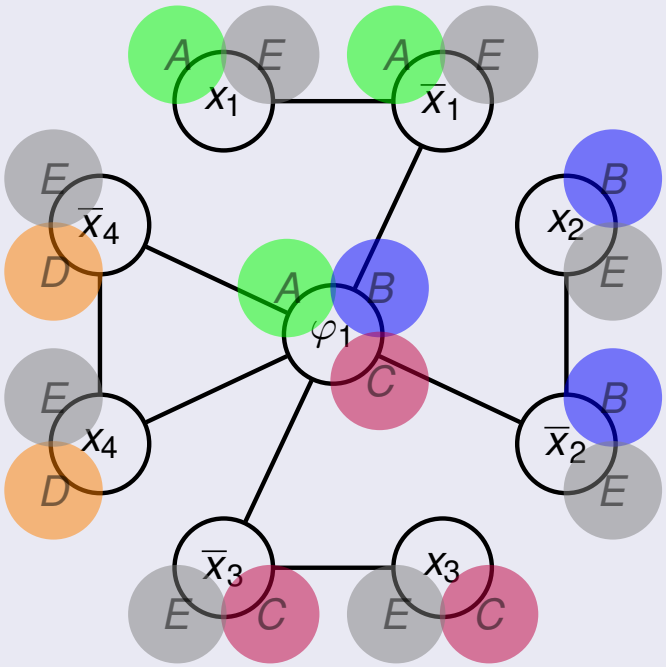
The node x_3 is coloured either by C (which stands for $x_3 = \mathit{true}$) or by E (for $x_3 = \mathit{false}$). The node $\overline{x}_3$ is coloured with the opposite colour.

GRAPH COLOURING

Proof (cont).

4. Add a node for every clause φ_i. For every x_j, connect φ_i with x_j if $x_j \notin \varphi_i$, and with $\overline{x}_j$ if $\neg x_j \notin \varphi_i$.

$$\underbrace{(x_1 \vee x_2 \vee x_3)}_{\varphi_1} \wedge \underbrace{(x_1 \vee \neg x_2 \vee \neg x_3)}_{\varphi_2} \wedge \underbrace{(\neg x_1 \vee x_2)}_{\varphi_3}$$



φ_1 can be coloured only if the colour of at least one of $\overline{x}_1, \overline{x}_2, \overline{x}_3$ is E.

$\rightarrow$ M is p-colourable iff

$$\varphi_1 \wedge \cdots \wedge \varphi_k$$

is satisfiable. □

SUBSET SUM

SUBSET SUM: Let S be a finite set of elements and w be the weight function $w : S \to \mathbb{Z}$. Is there a subset S' of elements of S, $S' \subseteq S$, s.t. the total weight of elements from S' is W, i.e.

$$\sum_{s \in S'} w(s) = W \ ?$$

Theorem

SUBSET SUM *is* **NP***-complete.*

Proof.

- SUBSET SUM is **NP**-hard — by reduction from 3-SAT. For a formula $\varphi_1 \wedge \cdots \wedge \varphi_k$ over variables $x_1, \ldots, x_n$, we set $S = \{t_1, \ldots, t_n, f_1, \ldots, f_n, c_1, \ldots, c_k, c'_1, \ldots, c'_k\}$ and assign values to w and W in the following way: *(next slide)*

SUBSET SUM

Proof (cont).

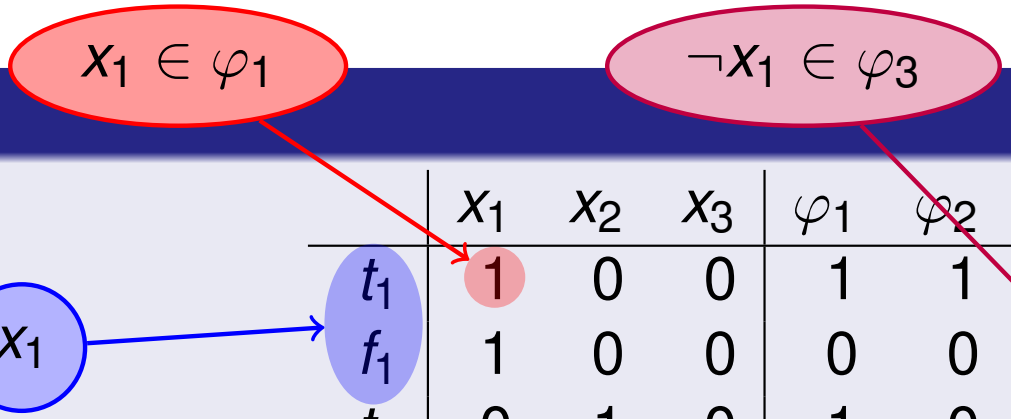
Assume the formula

$$\underbrace{(x_1 \vee x_2 \vee x_3)}_{\varphi_1} \wedge \underbrace{(x_1 \vee \neg x_2 \vee \neg x_3)}_{\varphi_2} \wedge \underbrace{(\neg x_1 \vee x_2)}_{\varphi_3}$$

- We consider decimal encoding of w and W of length $n + k$.
- Each variable x_i is assigned a pair of elements t_i and f_i.
- Each clause φ_j is assigned a pair of elements c_j and c'_j.

SUBSET SUM

Proof (cont).



	x_1	x_2	x_3	φ_1	φ_2	φ_3
t_1	1	0	0	1	1	0
f_1	1	0	0	0	0	1
t_2	0	1	0	1	0	1
f_2	0	1	0	0	1	0
t_3	0	0	1	1	0	0
f_3	0	0	1	0	1	0
c_1	0	0	0	1	0	0
c'_1	0	0	0	1	0	0
c_2	0	0	0	0	1	0
c'_2	0	0	0	0	1	0
c_3	0	0	0	0	0	1
c'_3	0	0	0	0	0	1
W	1	1	1	3	3	3

Assume the formula

$$\underbrace{(x_1 \vee x_2 \vee x_3)}_{\varphi_1} \wedge \underbrace{(x_1 \vee \neg x_2 \vee \neg x_3)}_{\varphi_2} \wedge \underbrace{(\neg x_1 \vee x_2)}_{\varphi_3}$$

- We consider decimal encoding of w and W of length $n + k$.
- Each variable x_i is assigned a pair of elements t_i and f_i.
- Each clause φ_j is assigned a pair of elements c_j and c'_j.

PARTITION

PARTITION: Let T be a finite set of elements and v be the weight function $v : T \to \mathbb{Z}$. Can T be partitioned into two sets T' and $T \setminus T'$ of equal total weight, i.e.

$$\sum_{t \in T'} v(t) = \sum_{t \in T \setminus T'} v(t) \; ?$$

Theorem

PARTITION *is* **NP***-complete.*

Proof.

- PARTITION is **NP**-hard — by reduction from SUBSET SUM. For the elements S, weight function w and target weight W, we set $T = S \cup \{z\}$ where $z \notin S$, and $v = w \cup \{z \mapsto (w(S) - 2W)\}$ where $w(S) = \sum_{s \in S} w(s)$. □

KNAPSACK

KNAPSACK: Let R be a finite set of elements, u be the weight function $u : R \to \mathbb{Z}$, and v be the value function $v : R \to \mathbb{Z}$. Is there a subset R' of elements of R, $R' \subseteq R$, s.t. the total weight of elements from R' is at most U and their total value is at least V, i.e.

$$\sum_{r \in R'} u(r) \leq U \ \wedge \ \sum_{r \in R'} v(r) \geq V \ ?$$

Theorem

KNAPSACK *is* **NP***-complete.*

Proof.

- KNAPSACK is **NP**-hard — by reduction from SUBSET SUM. For the elements S, weight function w and target weight W, we set $R = S$, $u = w$, $v = w$, $U = W$, and $V = W$. □