NP Completeness

Many problems are NP-complete; no polynomial-time solution is known for them, but the existence of such is not known impossible either.

Define problem classes $\mathcal{P}$ and $\mathcal{NP}$:

\[ \mathcal{P} = \{ L | \text{some deterministic TM recognizes } L \text{ in time limited by some polynomial } \} \]

\[ \mathcal{NP} = \{ L | \text{some non-deterministic TM recognizes } L \text{ in time limited by some polynomial } \} \]

Obviously $\mathcal{P} \subseteq \mathcal{NP}$, but does it hold that $\mathcal{P} = \mathcal{NP}$?

Example: $m \times m$ monkey puzzles

NP-complete problems are the "most difficult" ones within class $\mathcal{NP}$

Comparing difficulty of problems (within polynomial time):

Polynomial reduction

Language $A$ is polynomial-time reducible to $B$,

\[ A \leq_{P} B \]

if we can compute a function $f$ such that

\[ x \in A \iff f(x) \in B \]

in polynomial time

If problem $L$ can be solved in polynomial time, by

1. non-deterministically choosing a candidate solution, and
2. verifying it,

then $L$ belongs to class $\mathcal{NP}$

If $A \leq_{P} B$, then "$w \in A$?" is (within polynomial time) at most as difficult as the question "$w \in B$?":

Let $f$ be a polynomial reduction $A \leq_{P} B$

$\sim$ membership test for $A$:

\begin{verbatim}
procedure memberOfA(w) returns boolean
  w':= f(w);
  return memberOfB(w');
end procedure
\end{verbatim}

If $A \leq_{P} B$ and

1. $B \in \mathcal{P}$, then $A \in \mathcal{P}$;
2. $A \notin \mathcal{P}$, then $B \notin \mathcal{P}$.
Definition of NP-completeness

\( L \) is **NP-hard** \((\text{NP-vaikaa})\), if all languages of \( \mathcal{NP} \) are polynomially reducible to \( L \).

\( L \) is **NP-complete** \((\text{NP-täydellinen})\), if

1. \( L \in \mathcal{NP} \)
2. \( L \) is NP-hard

The question of efficient solvability of any NP-complete problem captures the famous \( \mathcal{P} \subseteq \mathcal{NP} \) problem:

**Theorem** Let \( L \) be a NP-complete problem. Then \( L \in \mathcal{P} \) \( \iff \mathcal{P} = \mathcal{NP} \)

**Proof:**

"\( \Leftarrow \)" If \( \mathcal{P} = \mathcal{NP} \), then \( L \in \mathcal{P} \)

"\( \Rightarrow \)" Assume \( L \in \mathcal{P} \). Let \( A \in \mathcal{NP} \). Since \( A \leq_p L \), the question \( \text{"} w \in A \text{"} \) can be solved in polynomial time as \( \text{"} f(w) \in L \text{"} \) \( \rightarrow A \in \mathcal{P} \)

How do we show problems NP-hard?

By reduction \textit{from} some NP-hard problem:

(\textbf{NB} direction!)

**Theorem** If \( L \) is NP-hard and \( L \leq_p L' \), then also \( L' \) is NP-hard.

**Proof.** Let \( A \in \mathcal{NP} \), and \( f \) reduction for \( A \leq_p L \) in time \( p(n) \)

Let \( g \) be reduction for \( L \leq_p L' \) in time \( q(n) \)

Construct \( f \circ g \):

\[
\begin{align*}
    w' := f(w); \\
    \text{return } g(w'); \\
\end{align*}
\]

Now \( f \circ g \) is a reduction \( A \leq_p L' \)

Observations on Satisfiability

**Example** \( w = (\neg x \vee y) \land \neg y \)

Value under a given truth assignment is easy to compute:

E.g., \( x \leftarrow F, y \leftarrow T \):

Satisfiability is straightforward to check:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>((\neg x \vee y) \land \neg y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

but tedious
Idea of the proof:

\[ A \in NP \iff A = L(M) \text{ for some NTM } M \text{ with polynomial time compl. } p(n) \]
- using these, construct polynomial reduction
  \[ w \mapsto \text{Accept}_A(w), \text{ such that } w \in L(M) \iff \text{Accept}_A(w) \in SAT \]

First show that we can restrict to 1-tape Turing machines:

**Lemma** Each language of class \(NP\) is accepted by some one-tape NTM in polynomial time

**Proof.** (Sketch)

Let \(M\) be TM with \(k\) tapes. Represent them in \(TM M'\) by a single tape with \(2k\) "tracks".
Tracks \(2i\) correspond to tapes of \(M'\), and tracks \(2i - 1\) to positions of tape heads:

\[ \begin{array}{l}
\text{tape 1} \\
\text{tape 2} \\
\text{tape 3} \\
\end{array} \]

If \(T_M(n) \leq p(n)\), then \(S_{M'}(n) \leq 2p(n)\)

\(M'\) simulates a single transition of \(M\):

1. Tape head initially at left, and state corresponds to state \(q\) of \(M\);
2. Scan the tape, moving to state that corresponds to symbols \(a_1, \ldots, a_k\) pointed by simulated tape heads;
3. Choose some transition in accordance to the transition relation \(\delta(q, a_1, \ldots, a_k)\) of \(M\).
4. Return tape head to left; On the way update the symbols at the simulated tape heads

One step of \(M\) is simulated by \(O(p(n))\) steps of \(M'\)

\(\sim M'\) works in time \(O(p(n)^2)\) \(\square\)

**Theorem** SAT is NP-complete
(S. Cook, 1971)

**Proof.** (Sketch)

1. SAT \(\in NP\): Guess a truth assignment and check whether it satisfies the input formula

2. NP-hardness: Let \(A \in NP\) be recognized with 1-tape NTM \(M\), \(T_M(n) = p(n)\)

Transform \(M\)'s input \(w\) in polynomial time to formula \(\text{Accept}_A(w)\), such that
\(w \in A \iff \text{Accept}_A(w) \in SAT\)

Formula \(\text{Accept}(w)\) describes accepting computations of \(M\) with input \(w\)
Let the states and the tape symbols of $M$ be $\{q_0, \ldots, q_k\}$ and $\{a_0, \ldots, a_m\}$ ($a_0 = b$).

Use different variables for denoting contents of tape squares, and the state + tape-head position, at different times $0, \ldots, p(n)$:

- $a_i^{s,t} \iff$ at time $t$, square $s$ contains symbol $a_i$.
- $q_j^{s,t} \iff$ at time $t$, tape-head is at square $s$ and $M$ is in state $q_j$.

The number of these variables is $O(p(n)^2)$.

Accept$(w)$ can, in polyn. time, be simplified into **Conjunctive Normal Form** (CNF)

$$D_1 \land \cdots \land D_k,$$

where each $D_i$ is a disjunction (a clause) of literals (variable or its negation).

$\Rightarrow$ **CNF-SAT** = {satisfiable CNF-formulas} is NP-complete, too.

(Often SAT $\equiv$ CNF-SAT)

**Example** $(a \lor b \lor \overline{c}) \land (\overline{a} \lor \overline{b}) \land (b \lor \overline{c}) \land (c)$

Each clause must become true, meaning that

some of its literals must become true

Let’s restrict (CNF-)SAT further:

3SAT = \{ $w$ | $w$ is satisfiable and in CNF-form, with exactly 3 literals in each clause \}

Example:

$$(a_1 \lor a_2) \land (\overline{a}_1 \lor \overline{a}_2 \lor a_3) \in$ CNF-SAT – 3SAT

An equivalent 3CNF-formula:

$$(a_1 \lor a_2 \lor b) \land (a_1 \lor a_2 \lor \overline{b}) \land (\overline{a}_1 \lor \overline{a}_2 \lor a_3)$$
Theorem 3SAT is NP-complete

Proof

(i) 3SAT $\in \mathcal{NP}$: Same as with SAT $\in \mathcal{NP}$

(ii) NP-hardness: CNF-SAT $\leq_p$ 3SAT by transforming too short or too long clauses $f_i$ of formula $w = f_1 \land \cdots \land f_m$ as follows:

$$f_i = (l_1) \sim (l_1 \lor b_1 \lor b_2) \land (l_1 \lor b_1 \lor \overline{b_2})$$

$$f_i = (l_1 \lor l_2) \sim (l_1 \lor l_2 \lor b_1) \land (l_1 \lor l_2 \lor \overline{b_1})$$

$$f_i = (l_1 \lor l_2 \lor l_3 \lor \cdots \lor l_n) \sim (l_1 \lor l_2 \lor \overline{b_2}) \land (l_2 \lor l_3 \lor \overline{b_3}) \land \cdots \land (l_{n-2} \lor l_{n-1} \lor \overline{l_n})$$

This can be done in linear time, and the new formula is satisfiable iff $w$ is $\square$

Other NP-complete problems

Colorability

Instance: Pair $(k, G)$, where $k$ is an integer and $G$ a graph

Accept the input if vertices of $G$ can be colored using $k$ colors without any two neighbors getting the same color

Example

3, $v_1, v_2, v_3, (v_1, v_2), (v_1, v_3), (v_2, v_3)$ $\in$ Colorability

2, $v_1, v_2, v_3, (v_1, v_2), (v_1, v_3), (v_2, v_3)$ $\not\in$ Colorability

Construct an "assignment subgraph":

For each clause $f_i = (l_1 \lor l_2 \lor l_3)$, construct a "clause subgraph":

10

17

18

19
Example: $G(w)$ for $w = (a \lor b \lor c) \land (\bar{a} \lor b \lor c)$:

$G(w)$ can be constructed in linear time.

Now $w$ is satisfiable iff $G(w)$ is 3-colorable:

a) If graph $G(w)$ has a 3-coloring, where $T = \text{color}(t)$, $F = \text{color}(f)$, $R = \text{color}(r)$, the color of each literal is either $T$ or $F$.

In each clause subgraph, the color of at least one literal can be seen to be $T$.

$\Rightarrow$ the formula is satisfied by assigning $\text{True}$ to the literals that are colored by $T$.

b) If formula $w$ is satisfiable, assign the color of nodes for $\text{True}$ literals to $T$ (and set color of others to $F$).

$\rightarrow$ In each clause subgraph, at least one literal node is colored $T$.

This can be extended into a full 3-coloring (where node $t$ gets color $T$).

Color minimization problem: Find out the chromatic number of a graph, that is, smallest number of colors to get different color to the end points of each edge.

Also optimization (or search) problems are called NP-hard, if their polynomial-time solution would imply $P = NP$.

Color minimization is obviously NP-hard: $(k, G) \in \text{Colorability} \iff$ the chromatic number of $G$ is at most $k$.

Conversely, an efficient algorithm for a decision problem often leads to an efficient solution of the corresponding optimization problem, too.

For example, to find out the chromatic number of graph $G$:

- $k := 0$;
- repeat
  - $k := k + 1$;
- until $(k, G) \in \text{Colorability}$;
- return $k$;

Exact cover problem

Input: Set $S_0$ and its subsets $S_1, S_2, \ldots , S_n$.

Accept the input, if we can pick $S_{i_1}, \ldots , S_{i_m}$ ($1 \leq i_1 < \ldots < i_m \leq n$), such that

$S_0 = S_{i_1} \cup S_{i_2} \cup \ldots \cup S_{i_m}$,

and $S_{i_j} \cap S_{i_k} = \emptyset$.

Example

$\{1, 2, 3, 4, 5\}, \{1, 3\}, \{2, 3, 4\}, \{2, 4\}, \{5\}$

$\in \text{ExactCover}$

$\{1, 2, 3, 4, 5\}, \{1, 3\}, \{2, 4\}, \{2, 4\}, \{3, 5\}$

$\notin \text{ExactCover}$
Theorem Exact Cover is NP-complete

Proof. Exact Cover ∈ \textit{NP}: Guess which subsets to include, and check

NP-hardness is shown by reduction

3-colorability \leq_p Exact Cover:
Let the instance of Colorability be

\[3, v_1, \ldots, v_n, (u_1, v_1), \ldots, (u_{m}, v_{m})\]

For each vertex \(v_i\) and color \(l \in \{1, 2, 3\}\), construct a set \(C_{v_i,l}\) as

\[\{v_i\} \cup \{[e,l] \mid \text{edge } e \text{ emanates from } v\}\]

\(S_0 \leftarrow \text{union of all these sets}\)

In addition, \(D_{el} = \{[e,l]\} \text{ for each } e \in E \text{ and } l \in \{1, 2, 3\}\)

This ExactCover instance can be constructed in polynomial time.

Now \(G\) is 3-colorable iff \(S_0\) has an exact cover using \(C_{el}\) and \(D_{el}\) sets:

3-colorability \Rightarrow Exact cover:
Assume that \(G\) has a 3-coloring

Then we get an exact cover by taking sets \(C_{v,\text{color}(v)}\) and those of \(D_{el}\) whose item \([e,l]\) is not included in \(\bigcup C_{v,\text{color}(v)}\):

Each member of \(S_0\) is covered

Singletons \(D_{el}\) do not overlap with other members of the cover

\(C_{u,\text{color}(u)}\) and \(C_{v,\text{color}(v)}\) can have a common member \([e,l]\) only if \(e = (u,v)\) and \(\text{color}(u) = l = \text{color}(v)\), but this is not allowed by the correct coloring.

\textbf{Knapsack problem}

Input: Positive integers \(s, i_1, \ldots, i_n\)

Accept the input, if one can choose \(1 \leq j_1 < j_2 < \cdots < j_k \leq n\) such that \(s = i_{j_1} + \cdots + i_{j_k}\)

Theorem Knapsack is NP-complete

Proof. (1) Knapsack ∈ \textit{NP}: Guess which numbers to pick, and check their sum

(2) \text{NP-hardness: ExactCover} \leq_p \text{Knapsack}:
Let an instance of ExactCover be \(S_0, S_1, \ldots, S_n\), where \(S_0 = \{a_1, \ldots, a_m\}\)

Represent each \(S_i\) as a \((n+1)\)-base number

\[s_i = d_{im}d_{i,m-1} \ldots d_{i1}, \text{ where}\]

\[d_{ij} = \begin{cases} 1 & \text{if } a_j \in S_i \\ 0 & \text{if } a_j \notin S_i \end{cases}\]
Now $S_0$ has an exact cover using $S_1, \ldots, S_n$ → a subsequence of $s_1, \ldots, s_n$ sums up to $s_0$:

**Exact cover ⇒ Knapsack:** If $S_{i_1}, \ldots, S_{i_k}$ is an exact cover of $S_0$ then obviously

$$s_{i_1} + \ldots + s_{i_k} = s_0$$

**Knapsack ⇒ Exact cover:** Assume that

$$s_{i_1} + \ldots + s_{i_k} = s_0$$

is a solution to the Knapsack instance

Numbers are $(n + 1)$-based and $k \leq n$ ⇒ no carry appears in the addition

Therefore, for each $j = 1, \ldots, m$,

$$s_{0j} = s_{i_{1j}} + \ldots + s_{i_{kj}} = 1$$

⇒ $s_{ij} = 1$ for exactly one $l \in \{1, \ldots, k\}$

⇒ $a_{ij} \in S_l$ for exactly one $l \in \{1, \ldots, k\}$

Thus $S_{i_1}, \ldots, S_{i_k}$ is an exact cover of $S_0$ \[\square\]

What about the strange base $(n + 1)$?

Numbers can be transformed to a different base (say, 10 or 2) in polynomial time

⇒ Knapsack is NP-complete also in ordinary $d$-base number systems

Unary representation is exponentially longer than $d$-based ($d > 1$), which may influence complexity

(Remember: wrt the length of input)

Example: Knapsack instance

$11, 5, 14, 4, 3$

using unary coding:

$1111111111, 11111, 11111111111111, 1111, 111$

Indeed, unary-coded Knapsack can be solved, using Dynamic Programming, in time $O(|w|^2)$

**Theorem** TSP is NP-complete

**Proof.** \(TSP \in \text{NP}^c\):

Guess the ordering, and check it’s OK

NP-hardness: Knapsack ≤_P TSP:

Let $k, i_1, \ldots, i_n$ be an instance of Knapsack

Construct a graph with nodes $\{v_0, \ldots, v_n+1\}$

and edges with weights

$$w(v_q, v_r) = \begin{cases} i_r & \text{if } q < r \leq n \\ 0 & \text{if } q > r \text{ or } r = n + 1 \end{cases}$$
Now Knapsack has a solution iff the graph contains a TSP route of length $k$:

1) Assume a TSP route of length $k$

The path enters each vertex once

$\Rightarrow k$ is sum of unique numbers from $i_1, \ldots, i_n$

2) Assume $i_{j_1} + \ldots + i_{j_m} = k$ ($j_1 < \ldots < j_m$)

Order vertices as follows:

$v_0, v_{j_1}, \ldots, v_{j_m}, v_{n+1}, v_{l_1}, \ldots, v_{l_p}$

where $i_1, \ldots, i_p$ are indices from

$\{1, 2, \ldots, n\} \setminus \{j_1, \ldots, j_m\}$

in descending order

This order is a TSP route of length $k$ \(\Box\)

On efficient solvability

Small difference in problem formulation can change complexity drastically:

| Unsolvable | NP-hard problem
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{P}$</td>
<td>Hitting problem</td>
</tr>
<tr>
<td>$\text{NP}$</td>
<td>Hitting's NP-complete problem</td>
</tr>
</tbody>
</table>

| $\text{P}$ | Exact poly-time algorithms
| $\text{NP}$ | Exact NP-hard
| $\Theta(n \log n)$ | $\Theta(n^2)$
| $\Theta(n)$ | $\Theta(n!)$

Exercise