ALBERT-LUDWIGS-
UNIVERSITÄT FREIBURG

Prof. Dr. Andreas Podelski
Matthias Heizmann
Christian Schilling

July 22nd-23rd, 2014

8. Lecture
Computer Science Theory

Chapter VI – Complexity (pp. 123-138)

§1 Computational complexity (pp. 123-126)

Before, we always abstracted away from runtime and memory usage. Now we consider these two measurements to compare problems (with focus on runtime).
There are two important bounds for algorithmic problems:

- *upper bound*, meaning that this problem is solvable within that bound

  proven by giving an algorithm which solves the problem in this bound

- *lower bound*, meaning that there are instances of this problem which cannot be solved more efficiently

  no general proof method

Our model for reasoning: Turing machines. We can use k-tape TMs to get more realistic results (a TM has no random access, opposite to a computer).

Definition 1.1 Let $f : \mathbb{N} \to \mathbb{N}$ be a function and τ be a non-deterministic TM with several tapes and input alphabet Σ.

- τ has the *time complexity* $f(n)$ if for every $w \in \Sigma^*$ of length n it holds: τ applied to the input w terminates for every possible computation in at most $f(n)$ steps.

(ii) τ has the *space complexity* $f(n)$ if for every $w \in \Sigma^*$ of length n holds: τ applied to the input w uses for every possible computation on every tape at most $f(n)$ fields.

Definition 1.2 Let $f : \mathbb{N} \to \mathbb{N}$.

$$\begin{aligned}
\texttt{DTIME}(f(n)) &= \{L \mid \text{there exists a \textit{deterministic} TM with several tapes} \\ &\qquad \text{which has time complexity } f(n) \text{ and accepts } L\} \\
\texttt{NTIME}(f(n)) &= \{L \mid \text{there is a \textit{non-deterministic} TM with several tapes} \\ &\qquad \text{which has time complexity } f(n) \text{ and accepts } L\} \\
\texttt{DSPACE}(f(n)) &= \{L \mid \text{there exists a \textit{deterministic} TM with several tapes} \\ &\qquad \text{which has space complexity } f(n)\text{, and accepts } L\} \\
\texttt{NSPACE}(f(n)) &= \{L \mid \text{there is a \textit{non-deterministic} TM with several tapes} \\ &\qquad \text{which has space complexity } f(n) \text{ and accepts } L\}
\end{aligned}$$

Because a TM can visit at most one new field on its tapes in each computational step, we have:

$$\texttt{DTIME}(f(n)) \begin{array}{c} \subseteq \\ \subseteq \end{array} \begin{array}{c} \texttt{DSPACE}(f(n)) \\ \\ \texttt{NTIME}(f(n)) \end{array} \begin{array}{c} \subseteq \\ \subseteq \end{array} \texttt{NSPACE}(f(n))$$

Definition 1.3 Let $g : \mathbb{N} \to \mathbb{N}$. We define:

$$\mathcal{O}(g(n)) = \{f : \mathbb{N} \to \mathbb{N} \mid \exists n_0, k \in \mathbb{N} : \forall n \geq n_0 : f(n) \leq k \cdot g(n)\}$$

$\mathcal{O}(g(n))$ is the class of all functions f which are *bounded* by a constant multiplied by g for sufficiently large values of n.

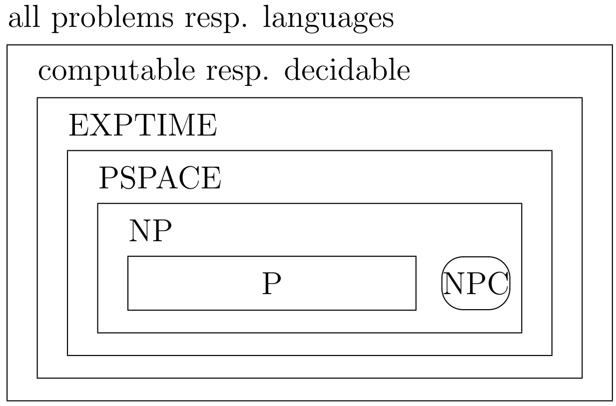
§2 The classes P and NP (pp. 127-132)

Definition 2.1

$$\begin{aligned}
P &= \bigcup_{p \text{ polynomial in } n} \texttt{DTIME}(p(n)) \\
NP &= \bigcup_{p \text{ polynomial in } n} \texttt{NTIME}(p(n)) \\
PSPACE &= \bigcup_{p \text{ polynomial in } n} \texttt{DSPACE}(p(n)) \\
NPSPACE &= \bigcup_{p \text{ polynomial in } n} \texttt{NSPACE}(p(n))
\end{aligned}$$

Theorem 2.2 $P \subseteq NP \subseteq PSPACE = NPSPACE \subseteq EXPTIME$

Open problem in computer science: Are the inclusions strict or does the equality hold? We only know $P \subsetneq EXPTIME$.



P: *Construct* the right solution deterministically and in polynomial time.

NP: *Guess* a solution proposal *non-deterministically* and then *verify / check* deterministically and in polynomial time whether this proposal is right.

Examples of problems from the class NP

(a) **Problem of Hamiltonian path**
Given: A finite graph with n vertices.
Question: Does the graph contain a Hamiltonian path, i.e., a path which visits each vertex exactly once?

(b) **Traveling salesman problem**
Given: A finite, complete graph with n vertices and associated with every edge a natural number (the length/weight/cost), as well as a number $k \in \mathbb{N}$.
Question: Is there a tour of length $\leq k$, i.e., is there a cycle in the graph with length $\leq k$ which visits each vertex at least once?

(3) **Satisfiability problem for Boolean expressions (shortly SAT)**
Given: A Boolean expression B, i.e., an expression which consists only of variables $x_1, x_2, \ldots, x_n$ connected by operators $\neg$ (*not*), $\wedge$ (*and*) and $\vee$ (*or*), as well as by brackets.
Question: Is B satisfiable, i.e., is there an assignment of 0 and 1 to the Boolean variables $x_1, x_2, \ldots, x_n$ in B such that B evaluates to 1?

While thinking about polynomial solutions for these problems, a subclass of NP was found, namely the class NPC of the so called *NP-complete* problems. These are the hardest problems in NP and if one of them is found to be

polynomial, we immediately have a procedure to solve *all* problems in NP polynomially. This is why it is very unlikely that $P = NP$.
Today more than 1000 problems are known to be in NPC. Many of them are practically relevant.

Definition 2.3 Let $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ be languages. Then L_1 is called *polynomially-time reducible to* L_2, shortly

$$L_1 \leq_p L_2,$$

if there is a function $f : \Sigma_1^* \to \Sigma_2^*$ which is total and computable with a polynomial time complexity, such that for all $w \in \Sigma_1^*$ it holds that

$$w \in L_1 \Leftrightarrow f(w) \in L_2.$$

We also say: $L_1 \leq_p L_2$ *using* f.

Definition 2.4 A language L_0 is called *NP-hard* if for all $L \in NP$ it holds that $L \leq_p L_0$.
A language L_0 is called *NP-complete* if $L_0 \in NP$ and L_0 is NP-hard.

Note that having only one of these properties is not helpful:

- If $L_0 \in NP$, then it might also be that $L_0 \in P$ (assuming $P \neq NP$).
- If L_0 is NP-hard, then it might be of exponential or worse complexity.

Lemma 2.5 Let $L_1 \leq_p L_2$. Then it holds that:

(a) If $L_2 \in P$ holds, then $L_1 \in P$ holds as well.

(b) If $L_2 \in NP$ holds, then $L_1 \in NP$ holds as well.

(c) If L_1 is NP-complete and $L_2 \in NP$ holds, then L_2 is also NP-complete.

Corollary 2.6 Let L be an NP-complete language. Then it holds that $L \in P \Leftrightarrow P = NP$.

Example reduction We reduce the Hamiltonian path problem (HPP) to the traveling salesman problem (TSP). We show this by an intermediate reduction for the Hamiltonian cycle/circuit problem (HCP). HCP is almost the same as HPP, only that we want a cycle now (i.e., the path should start and end at the same vertex, i.e., one vertex is visited twice). Obviously, HCP is also an NP problem (why?).

(a) Show HPP $\leq_p$ HCP.

<u>Construction</u>:

input: finite graph $G = (V, E)$

output: finite graph $G' = (V', E')$ with

- $V' = V \cup \{v_0\}, v_0 \notin V$, so we have the old vertices plus one extra vertex v_0
- $E' = E \cup \{(v, v_0) \mid v \in V\}$, so we have the old edges plus an edge from every old vertex to v_0.

This construction is total (recognizing a proper input is simple) and computable in polynomial time (we need to add one vertex and $|V|$ edges, possible in $O(|V| + |E|)$).

We need to show that $G \in$ HPP $\Leftrightarrow (G', k) \in$ HCP.

"$\Rightarrow$" If there is a Hamiltonian path in G, say, from v_1 to v_2, then we can close the cycle by taking the edge (v_2, v_0) to v_0, followed by taking the edge (v_1, v_0) to v_1 in G'.

"$\Leftarrow$" If there is a Hamiltonian cycle G', then every vertex is visited exactly once, especially v_0. We find a Hamiltonian path in G by removing those two transitions from and to v_0 on the cycle.

(b) Show HCP $\leq_p$ TSP.

<u>Construction</u>:

input: finite graph $G = (V, E)$

output: tuple (G', k), where

- $G' = (V, E')$ is a complete graph with E' has all edges from E with weight 1 and all possible edges not in E with weight 2
- $k = |V|$ is a natural number equal to the number of vertices.

This construction is total (recognizing a proper input is simple) and computable in polynomial time (there are $\mathcal{O}(|V|^2)$ possible edges).

We need to show that $G \in$ HCP $\Leftrightarrow (G', k) \in$ TSP.

"$\Rightarrow$" If there is a Hamiltonian cycle in G, then there is a tour in G' of weight k, namely the same cycle.

"$\Leftarrow$" If there is a tour in G' of weight k, it means that every edge has weight 1. This means these edges were already present in G, so we have a Hamiltonian cycle in G, namely the same as the tour.

(c) We have shown HPP $\leq_p$ HCP $\leq_p$ TSP. Because HPP is known to be NP-complete and thus NP-hard, we have shown that HCP and TSP are NP-hard. Since HCP and TSP are NP problems, they are even NP-complete.

Some NP-complete problems can be slightly modified and then polynomially solved. For instance, the Eulerian path problem asks for a path through a graph which visits each edge exactly once (compare to the Hamiltonian path problem). This is polynomially solvable (linear in the size of the graph).

A typical way to tackle NP-complete problems in practice is finding good heuristics. Remember that lower bounds only have to hold for *some* instances, so many instances might be simple to solve.
There exist important NP-complete problems for which very good approximation algorithms (e.g., the result is always at most 10% away from the optimum) or probability algorithms (e.g., the result is correct in 90% of all cases) are known.

§3 The satisfiability problem for Boolean expressions (pp. 133-138)

We skip this part in the interest of time.

Summary: The "mother of all NP-complete problems", SAT, is proven to be NP-complete. The proof is not very hard, but very technical.