Advanced Topics in Theoretical Computer Science

Part 5: Complexity (Part III)

6.02.2014

Viorica Sofronie-Stokkermans

Universität Koblenz-Landau

e-mail: sofronie@uni-koblenz.de

Contents

- Recall: Turing machines and Turing computability
- Register machines (LOOP, WHILE, GOTO)
- Recursive functions
- The Church-Turing Thesis
- Computability and (Un-)decidability
- Complexity

Until now

- P, NP, PSPACE

  $P \subseteq NP \subseteq PSPACE$

- closure properties

- it is not known whether:

  $P = NP$, $NP = \text{co-}NP$, $P = PSPACE$, $NP = PSPACE$

- How to show that a certain problem is in a certain complexity class?

  Reductions

Reduction

Definition (Polynomial time reducibility)

Let L_1, L_2 be languages.
L_2 is polynomial time reducible to L_1 (notation: $L_2 \preceq_{\text{pol}} L_1$)
if there exists a polynomial time bounded DTM, which for every input w computes an output $f(w)$ such that

$$w \in L_2 \text{ if and only if } f(w) \in L_1$$

Lemma (Polynomial time reduction)

- Let L_2 be polynomial time reducible to L_1 ($L_2 \preceq_{\text{pol}} L_1$). Then:

  If $L_1 \in NP$ then $L_2 \in NP$.

  If $L_1 \in P$ then $L_2 \in P$.

- The composition of two polynomial time reductions is again a polynomial time reduction.

Complete and hard problems

Definition (NP-complete, NP-hard)

- A language L is NP-hard (NP-difficult) if every language L' in NP is reducible in polynomial time to L.
- A language L is NP-complete if:
  – $L \in NP$
  – L is NP-hard

Definition (PSPACE-complete, PSPACE-hard)

- A language L is PSPACE-hard (PSPACE-difficult) if every language L' in PSPACE is reducible in polynomial time to L.
- A language L is PSPACE-complete if:
  – $L \in PSPACE$
  – L is PSPACE-hard

Examples of NP-complete problems

Examples of NP-complete problems:

1. Is a logical formula satisfiable? (SAT, 3-CNF) last time
2. Does a graph contain a clique of size k? (Clique of size k)
3. Rucksack problem (knapsack)
4. Can a graph be colored with three colors? (3-colorability)
5. Is a (un)directed graph hamiltonian? (Hamiltonian circle)
6. Has a set of integers a subset with sum x? (subset sum)
7. Multiprocessor scheduling

Examples of NP-complete problems

Examples of NP-complete problems:

1. Is a logical formula satisfiable? (SAT)
2. Does a graph contain a clique of size k? last time
3. Rucksack problem
4. Is a (un)directed graph hamiltonian?
5. Can a graph be colored with three colors?
6. Multiprocessor scheduling

Examples of NP-complete problems

Examples of NP-complete problems:

1. Is a logical formula satisfiable? (SAT)
2. Does a graph contain a clique of size k?
3. Rucksack problem today
4. Can a graph be colored with three colors?
5. Is a (un)directed graph hamiltonian?
6. Multiprocessor scheduling

Examples of NP-complete problems

Definition (Rucksack problem)

A rucksack problem consists of:

- n objects with weights $a_1, \ldots, a_n$
- a maximum weight b

The rucksack problem is solvable if there exists a subset of the given objects with total weight b.

$$\text{Rucksack} = \{(b, a_1, \ldots, a_n) \in \mathbb{N}^{n+1} \mid \exists I \subseteq \{1, \ldots, n\} \ s.t. \ \sum_{i \in I} a_i = b\}$$

Examples of NP-complete problems

Theorem Rucksack is NP-complete.

Proof: (1) Rucksack is in NP: We guess I and check whether $\sum_{i \in I} a_i = b$

(2) Rucksack is NP-hard: We show that 3-CNF-SAT $\prec_{\text{pol}}$ Rucksack.

Construct $f : \text{3-CNF} \rightarrow \mathbb{N}^*$ as follows.

Consider a 3-CNF formula $F = (L_1^1 \vee L_2^1 \vee L_3^1) \wedge \dots \wedge (L_1^m \vee L_2^m \vee L_3^m)$

$f(F) = (b, a_1, \dots, a_n)$ where:

(i) a_i encodes which atom occurs in which clause as follows:
p_i positive occurrences; n_i negative occurrences (numbers with $n + m$ positions)

- first m digits of p_i: p_{i_j} how often i-th atom occurs positively in j-th clause
- first m digits of n_i: n_{i_j} how often i-th atom occurs negatively in j-th clause

- last n digits of p_i, n_i: p_{i_j}, n_{i_j} which atom is referred by p_i
  p_i, n_i contain 1 at position $m + i$ and 0 otherwise.

Example

Let the set Prop of propositional variables consist of $\{x_1, x_2, x_3, x_4, x_5\}$.

$F: \quad (x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee x_2 \vee \neg x_5) \wedge (\neg x_3 \vee \neg x_1 \vee x_4)$

$$
\begin{array}{llll}
p_1 = 100\ 10000 & \qquad & n_1 = 001\ 10000 \\
p_2 = 020\ 01000 & & n_2 = 100\ 01000 \\
p_3 = 000\ 00100 & & n_3 = 001\ 00100 \\
p_4 = 101\ 00010 & & n_4 = 000\ 00010 \\
p_5 = 000\ 00001 & & n_5 = 010\ 00001
\end{array}
$$

Satisfying assignment: $\mathcal{A}(x_1) = \mathcal{A}(x_2) = \mathcal{A}(x_5)$ and $\mathcal{A}(x_3) = \mathcal{A}(x_4) = 0$.

$$p_1 + p_2 + p_5 + n_3 + n_4 = \quad \underbrace{121}_{\substack{\text{all digits } \leq 3 \\ \text{because 3 lit./clause}}} \quad \underbrace{11111}_{\substack{\text{all 1} \\ \text{all atoms considered}}}$$

Examples of NP-complete problems

Proof: (ctd.) If we have a satisfying assignment $\mathcal{A}$, we take for every propositional variable x_i mapped to 0 the number n_i and for every propositional variable x_i mapped to 1 the number p_i.

The sum of these numbers is $b_1 \dots b_m \underbrace{1 \dots 1}_{n \text{ times}}$ with $b_i \leq 3$,

so $b_1 \dots b_m \underbrace{1 \dots 1}_{n} < \underbrace{4 \dots 4}_{m} \underbrace{1 \dots 1}_{n}$

Let $b := \underbrace{4 \dots 4}_{m} \underbrace{1 \dots 1}_{n}$. We choose $\{a_1, \dots, a_k\} = \{p_1, \dots, p_n\} \cup \{n_1, \dots, n_n\} \cup C$.

The role of the numbers in $C = \{c_1, \dots, c_m, d_1, \dots, d_m\}$ is to make the sum of the a_is equal to b: $c_{i_j} = 1$ iff $i = j$; $d_{i_j} = 2$ iff $i = j$ (they are zero otherwise).

$f(F) \in$ Rucksack iff a subset I of $\{a_1, \dots, a_k\}$ adds up to b

iff a subset I of $\{p_1, \dots, p_n\} \cup \{n_1, \dots, n_n\}$ adds up to $b_1 \dots b_m 1 \dots 1$

iff for a subset I of $\{p_1, \dots, p_n\} \cup \{n_1, \dots, n_n\}$ there exists an assignment

iff $\mathcal{A}$ with $\mathcal{A}(P_i) = 1$(*resp.* 0) iff p_i(*resp.* n_i) occurs in I iff F satisfiable

Summary

Examples of NP-complete problems:

1. Is a logical formula satisfiable? (SAT)
2. Does a graph contain a clique of size k?
3. Rucksack problem
4. Can a graph be colored with three colors?
5. Is a (un)directed graph hamiltonian?
6. Multiprocessor scheduling

Summary

Examples of NP-complete problems:

1. Is a logical formula satisfiable? (SAT)
2. Does a graph contain a clique of size k?
3. Rucksack problem
4. Can a graph be colored with three colors?
5. Is a (un)directed graph hamiltonian?
6. Multiprocessor scheduling

Examples of NP-complete problems

Definition (k-colorability) A undirected graph is k-colorable if every node can be colored with one of k colors such that nodes connected by an edge have different colors.

$L_{\mathsf{Color_k}}$: the language consisting of all undirected graphs which are colorable with at most k colors.

Examples of NP-complete problems

The k-colorability is NP complete

Proof: Exercise. *Hint:*

(1) Prove that the problen is in NP.

(2) Let $F = C_1 \wedge \cdots \wedge C_k$ in 3-CNF containing propositional variables $\{x_1, \ldots, x_m\}$.

Let $G = (V, E)$ be an undirected graph, that is defined as follows:

$$V = \{C_1, \ldots, C_k\} \cup \{x_1, \ldots, x_m\} \cup \{\overline{x_1}, \ldots, \overline{x_m}\} \cup \{y_1, \ldots, y_m\}$$

$$\begin{aligned} E = & \{(x_i, \overline{x_i}), (\overline{x_i}, x_i) \mid i \in \{1, ..., m\}\} \cup \{(y_i, y_j) \mid i \neq j\} \cup \\ & \{(y_i, x_j), (x_j, y_i) \mid i \neq j\} \cup \{(y_i, \overline{x_j}), (\overline{x_j}, y_i) \mid i \neq j\} \cup \\ & \{(C_i, x_j), (x_j, C_i) \mid x_j \text{ not in } C_i\} \cup \{(C_i, \overline{x_j}), (\overline{x_j}, C_i) \mid \overline{x_j} \text{ not in } C_i\} \end{aligned}$$

Use G to prove 3-CNF-SAT $\preceq_{\text{pol}}$ k-colorability.

Examples of NP-complete problems

Examples of NP-complete problems:

1. Is a logical formula satisfiable? (SAT)
2. Does a graph contain a clique of size k?
3. Rucksack problem
4. Can a graph be colored with three colors?
5. Is a (un)directed graph hamiltonian?
6. Multiprocessor scheduling

Examples of NP-complete problems

Definition (Hamiltonian-cycle)

Path along the edges of a graph which visits every node exactly once.

Examples of NP-complete problems

Definition (Hamiltonian-cycle)
Path along the edges of a graph which visits every node exactly once and is a cycle.

$L_{\text{Ham,undir}}$: the language consisting of all undirected graphs which contain a Hamiltonian cycle

Examples of NP-complete problems

Definition (Hamiltonian-cycle)
Path along the edges of a graph which visits every node exactly once.

$L_{\text{Ham,undir}}$: the language consisting of all undirected graphs which contain a Hamiltonian cycle

$L_{\text{Ham,dir}}$: the language consisting of all directed graphs which contain a Hamiltonian cycle

NP-completeness: again reduction from 3-CNF-SAT.

Examples of NP-complete problems

Theorem. The problem whether a directed graph contains a Hamiltonian cycle is NP-complete.

Proof. (1) The problem is in NP: Guess a permutation of the nodes; check that they form a Hamiltonian cycle (in polynomial time).

(2) The problem is NP-hard. Reduction from 3-CNF-SAT.

$F = (L_1^1 \vee L_2^1 \vee L_3^1) \wedge \cdots \wedge (L_1^k \vee L_2^k \vee L_3^k)$

Construct $f(F) = G$ such that G contains a Hamiltonian cycle iff F satisfiable.

The details can be found in Erk & Priese, "Theoretische Informatik", p.466-471.

Examples of NP-complete problems

Examples of NP-complete problems:

1. Is a logical formula satisfiable? (SAT)
2. Does a graph contain a clique of size k?
3. Rucksack problem
4. Can a graph be colored with three colors?
5. Is a (un)directed graph hamiltonian?
6. Multiprocessor scheduling

Examples of NP-complete problems

Definition (Multiprocessor scheduling problem)

A scheduling problem consists of:

- n processes with durations $t_1, \ldots, t_n$
- m processors
- a maximal duration (deadline) D

The scheduling problem has a solution if there exists an distribution of processes on the processors such that all processes end before the deadline D.

L_{schedule} : the language consisting of all solvable scheduling problems

Other complexity classes

Co-NP

co-NP is the class of all laguages for which the complement is in NP

Example:

$L_{\text{tautologies}} = \{w \mid w \text{ is a tautology in propositional logic}\}$

Theorem. $L_{\text{tautologies}}$ is in co-NP.

Proof. The complement of $L_{\text{tautologies}}$ is the set of formulae whose negation is satisfiable, thus in NP.

PSPACE

Definition (PSPACE-complete, PSPACE-hard)
A language L is PSPACE-hard (PSPACE-difficult) if every language L' in PSPACE is reducible in polynomial time to L.

A language L is PSPACE-complete if:
- $L \in PSPACE$
- L is PSPACE-hard

Quantified Boolean Formulae

Syntax: Extend the syntax of propositional logic by allowing quantification over propositional variables.

Semantics:
$(\forall P)F \mapsto F[P \mapsto 1] \wedge F[P \mapsto 0]$
$(\exists P)F \mapsto F[P \mapsto 1] \vee F[P \mapsto 0]$

PSPACE

A fundamental PSPACE problem was identified by Stockmeyer and Meyer in 1973.

Quantified Boolean Formulas (QBF)

Given: A well-formed quantified Boolean formula $F = (Q_1 x_1) \ldots (Q_n x_n) E(x_1, \ldots, x_n)$

where E is a Boolean expression containing the variables $x_1, \ldots, x_n$ and Q_i is $\exists$ or $\forall$.

Question: Is F true?

(Does it evaluate to 1 if we use the evaluation rules above?)

PSPACE

Theorem QBF is PSPACE complete

Proof (Idea only)

(1) QBF is in PSPACE: we can try all possible assignments of truth values one at a time and reusing the space (2^n time but polynomial space).

(2) QBF is PSPACE complete. We can show that every language L' in PSPACE can be polymomially reduced to QBF using an idea similar to that used in Cook's theorem (we simulate a polynomial space bounded computation and not a polynomial time bounded computation).

The structure of PSPACE

The structure of PSPACE

… Beyond NP

The structure of PSPACE

Extend the notion of polynomial reducibility:

Nondeterministic Turing Machine with an oracle: NTM + oracle tape

– makes initial guess

– consult an oracle

Informally: NOTM for problem P: nondeterministic algorithm with a subroutine for P.

The structure of PSPACE

Extend the notion of polynomial reducibility:

Nondeterministic Turing Machine with an oracle: NTM + oracle tape

– makes initial guess

– consult an oracle

Informally: NOTM for problem P: nondeterministic algorithm with a subroutine for P.

defines a so-called (polynomial time) nondeterministic Turing reduction

The structure of PSPACE

The polynomial hierarchy

$P^Y = \{L \mid$ there exists a language $L' \in Y$ such that $L \preceq_{\text{pol}} L'\}$

$NP^Y = \{L \mid$ there exists a language $L' \in Y$ such that there exists a nondeterministic Turing reduction from L to $L'\}$

$$\Sigma_0^p = \Pi_0^p = \Delta_0^p = P.$$

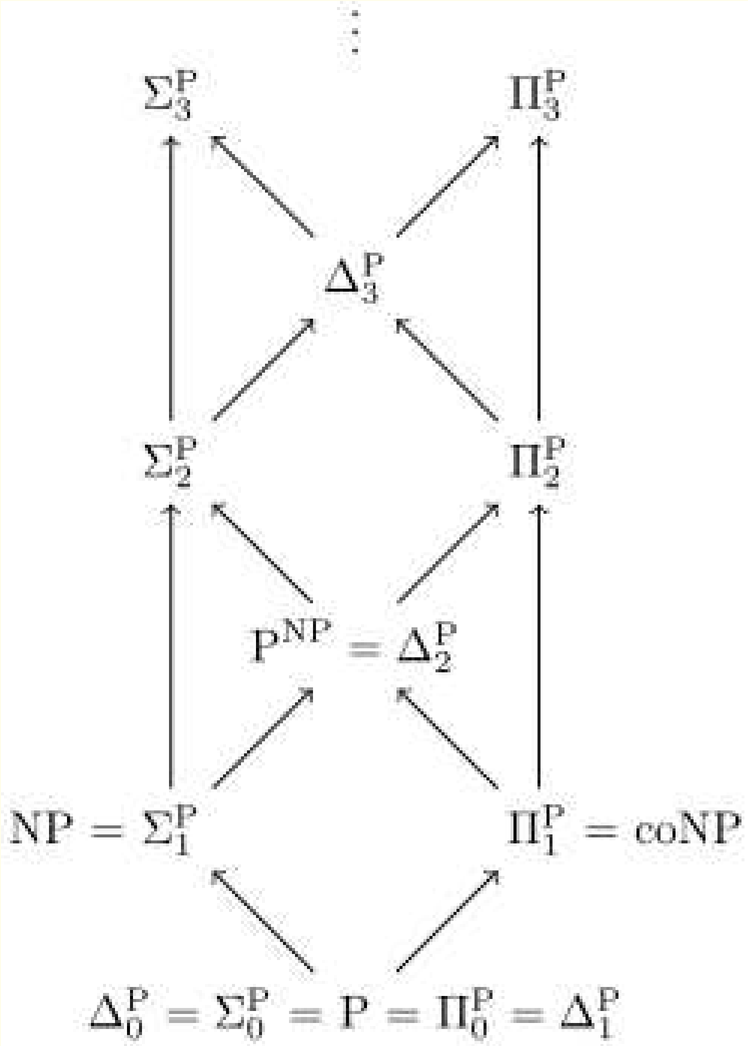
$$\Delta_{k+1}^p = P^{\Sigma_k^p}$$
$$\Sigma_{k+1}^p = NP^{\Sigma_k^p}$$
$$\Pi_{k+1}^p = \text{co-}NP^{\Sigma_k^p}$$

$\Pi_1^p = \text{co-NP}^P = \text{co-NP}$; $\Sigma_1^p = NP^P = NP$; $\Delta_1^p = P^P = P$.

$\Delta_2^p = P^{NP}$; $\Sigma_2^p = NP^{NP}$

The structure of PSPACE

PSPACE



The structure of PSPACE

A complete problem for Σ_k^{P} is satisfiability for quantified Boolean formulas with k alternations of quantifiers which start with an existential quantifier sequence (abbreviated QBF_k or $QSAT_k$).

(The variant which starts with $\forall$ is complete for Π_k^{P}).

Beyond PSPACE

EXPTIME, NEXPTIME

DEXPTIME, NDEXPTIME

EXPSPACE, ...

Discussion

- In practical applications, for having efficient algorithms polynomial solvability is very important; exponential complexity inacceptable.
- Better hardware is no solution for bad complexity

Question which have not been clarified yet:

- Does parallelism/non-determinism make problems tractable?
- Any relationship between space complexity and run time behaviour?

Other directions in complexity

Pseudopolynomial problems

Approximative and probabilistic algorithms

Motivation

Many important problems are difficult (undecidable; NP-complete; PSPACE complete)

- **Undecidable:** validity of formulae in FOL; termination, correctness of programs
- **NP-complete:** SAT, Scheduling
- **PSPACE complete:** games, market analyzers

Motivation

Possible approaches:

- Heuristic solutions:
  - use knowledge about the structure of problems in a specific application area;
  - renounce to general solution in favor of a good "average case" in the specific area of applications.
- Approximation: approximative solution
  - Renounce to optimal solution in favor of shorter run times.
- Probabilistic approaches:
  - Find correct solution with high probability.
  - Renounce to sure correctness in favor of shorter run times.

Approximation

Many NP-hard problems have optimization variants

- Example: Clique: Find a possible greatest clique in a graph

... but not all NP-difficult problems can be solved approximatively in polynomial time:

- Example: Clique: Not possible to find a good polynomial approximation (unless $\mathsf{P} = \mathsf{NP}$)

Probabilistic algorithms

Idea

- Undeterministic, random computation
- Goal: false decision possible but not probable
- The probability of making a mistake reduced by repeating computations
- 2^{-100} below the probability of hardware errors.

Probabilistic algorithms

Example: probabilistic algorithm for 3-Clique

NB: 3-Clique is polynomially solvable (unlike Clique)

Given: Graph $G = (V, E)$

Repeat the following k times:

- Choose randomly $v_1 \in V$ and $\{v_2, v_3\} \in E$
- Test if v_1, v_2, v_3 build a clique.

Error probability:

$k = (|E| \cdot |V|)/3$: Error probability < 0.5

$k = 100(|E| \cdot |V|)/3$: Error probability $< 2^{-100}$

Overview

- Register machines (LOOP, WHILE, GOTO)
- Recursive functions
- The Church-Turing Thesis
- Computability and (Un-)decidability
- Complexity
- Other computation models

Other computation models

- Variations of register machines (one register; two registers)
- Variations of TM; links with register machines
- Reversible computations: e.g. chemical reversibility or reversibility as in physics
- DNA Computing and Splicing
  Computing machines consisting from enzymes and molecules

Other computation models

- Variations of register machines (one register; two registers)
- Variations of TM; links with register machines
- Reversible computations: chemical and psysichal reversibility
- DNA Computing and Splicing
  Computing machines consisting from enzymes and molecules

Variants of automata

- Tree automata
- Automata over infinite words

Variants of automata

Tree automata

Like automata, but deal with tree structures, rather than the strings.

Tree automata are an important tool in computer science:

- compiler construction
- automatic verification of cryptographic protocols.
- processing of XML documents.

Variants of automata

Automata on infinite words (or more generally: infinite objects)

ω-Automata (Büchi automata, Rabin automata, Streett automata, parity automata and Muller automata)

- run on infinite, rather than finite, strings as input.
- Since ω-automata do not stop, they have a variety of acceptance conditions rather than simply a set of accepting states.

Applications: Verification, temporal logic

Look forward

Next semester:

- Seminar: Decision procedures and applications $\mapsto$ emphasis on decidability and complexity results for various application areas.
- Lecture: Formal verification and specification

Various possibilities for BSc/MSc thesis and Forschungspraktika.