Advanced Topics in Theoretical Computer Science

Part 5: Complexity (Part 2)

25.01.2023

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

P, NP, PSPACE

Definition

$$\begin{aligned} P &= \bigcup_{i\geq 1} DTIME(n^i) \\ NP &= \bigcup_{i\geq 1} NTIME(n^i) \\ PSPACE &= \bigcup_{i\geq 1} DSPACE(n^i) \end{aligned}$$

P, NP, PSPACE

Definition

$$\begin{array}{rcl} P & = & \bigcup_{i\geq 1} DTIME(n^i) \\ NP & = & \bigcup_{i\geq 1} NTIME(n^i) \\ PSPACE & = & \bigcup_{i\geq 1} DSPACE(n^i) \end{array}$$

Lemma $NP \subseteq \bigcup_{i\geq 1} DTIME(2^{O(n^d)})$

Proof: Follows from the fact that if L is accepted by a $f(n)$-time bounded NTM then L is accepted by an $2^{O(f(n))}$-time bounded DTM, hence for every $d \geq 1$ we have:

$$NTIME(n^d) \subseteq DTIME(2^{O(n^d)})$$

P, NP, PSPACE

$$\begin{array}{lcl} P & = & \bigcup_{i\geq 1} DTIME(n^i) \\ NP & = & \bigcup_{i\geq 1} NTIME(n^i) \\ PSPACE & = & \bigcup_{i\geq 1} DSPACE(n^i) \\ NP & \subseteq & \bigcup_{i\geq 1} DTIME(2^{O(n^d)}) \end{array}$$

Intuition

- Problems in P can be solved efficiently; those in NP can be solved in exponential time
- PSPACE is a very large class, much larger that P and NP.

Complexity classes for functions

Definition

A function $f : \mathbb{N} \to \mathbb{N}$ is in P if there exists a DTM M and a polynomial $p(n)$ such that for every n the value $f(n)$ can be computed by M in at most $p(\text{length}(n))$ steps.

Here $\text{length}(n) = \log(n)$: we need $\log(n)$ symbols to represent (binary) the number n.

The other complexity classes for functions are defined in an analogous way.

Relationships between complexity classes

Question:

Which are the links between the complexity classes P, NP and PSPACE?

Relationships between complexity classes

Question:

Which are the links between the complexity classes P, NP and PSPACE?

$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE}$

Complexity classes

How do we show that a certain problem is in a certain complexity class?

Reduction to a known problem

We need one problem we can start with! (for NP: SAT)

Complexity classes

Can we find in NP problems which are the most difficult ones in NP?

Complexity classes

Can we find in NP problems which are the most difficult ones in NP?

Answer

There are various ways of defining “the most difficult problem”.

They depend on the notion of reducibility which we use.

For a given notion of reducibility the answer is YES.

Such problems are called complete in the complexity class with respect to the notion of reducibility used.

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

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

Reduction

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.

Proof: Assume $L_1 \in P$. Then there exists $k \geq 1$ such that L_1 is accepted by n^k-time bounded DTM M_1.

Since $L_2 \preceq_{\text{pol}} L_1$ there exists a polynomial time bounded DTM M_f, which for every input w computes an output $f(w)$ such that $w \in L_2$ if and only if $f(w) \in L_1$.

Let $M_2 = M_f M_1$. Clearly, M_2 accepts L_2. We have to show that M_2 is polynomial time bounded. $w \mapsto M_f$ computes $f(w)$ (pol.size) $\mapsto M_1$ decides if $f(w) \in L_1$ (polynomially many steps)

NP

Theorem (Characterisation of NP)

A language L is in NP if and only if there exists a language L' in P and a $k \geq 0$ such that for all $w \in \Sigma^*$:

$$w \in L \text{ iff } \quad \text{there exists } c : \langle w, c \rangle \in L' \text{ and } |c| < |w|^k$$

c is also called witness or certificate for w in L.

A DTM which accepts the language L' is called verifier.

Important

A decision procedure is in NP iff every "Yes" instance has a short witness

(i.e. its length is polynomial in the length of the input)

which can be verified in polynomial time.

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

Complete and hard problems

Remarks:

- If we can prove that at least one NP-hard problem is in P then P $=$ NP
- If P $\neq$ NP then no NP complete problem can be solved in polynomial time

Open problem: Is P $=$ NP? (Millenium Problem)

How to show that a language L is NP-complete?

1. Prove that $L \in NP$
2. Find a language L' known to be NP-complete and reduce it to L

Often used: the SAT problem (Proved to be NP-complete by S. Cook)

$$L' = L_{\mathsf{sat}} = \{w \mid w \text{ is a satisfiable formula of propositional logic}\}$$

Stephen Cook

Stephen Arthur Cook (born 1939)

- Major contributions to complexity theory.
  Considered one of the forefathers of computational complexity theory.
- 1971 ‘The Complexity of Theorem Proving Procedures’
  Formalized the notions of polynomial-time reduction and NP-completeness, and proved the existence of an NP-complete problem by showing that the Boolean satisfiability problem (SAT) is NP-complete.
- Currently University Professor at the University of Toronto
- 1982: Turing award for his contributions to complexity theory.

Cook's theorem

Theorem $SAT = \{w \mid w \text{ is a satisfiable formula of propositional logic}\}$ is NP-complete.

Cook's theorem

Theorem $SAT = \{w \mid w \text{ is a satisfiable formula of propositional logic}\}$ is NP-complete.

Proof (Idea)

To show: (1) $SAT \in NP$

(2) for all $L \in NP$, $L \preceq_{\text{pol}} SAT$

Cook's theorem

Theorem $SAT = \{w \mid w \text{ is a satisfiable formula of propositional logic}\}$ is NP-complete.

Proof (Idea)

To show: (1) $SAT \in NP$

(2) for all $L \in NP$, $L \preceq_{\text{pol}} SAT$

(1) Construct a k-tape NTM M which can accept SAT in polynomial time:

$w \in \Sigma_{PL}^*$ $\mapsto$ M does not halt if $w \not\in SAT$

M finds in polynomial time a satisfying assignment

(a) scan w and see if it a well-formed formula; collect atoms $\mapsto O(|w|^2)$

(b) if not well-formed: inf.loop; if well-formed M guesses a satisfying assignment $\mapsto O(|w|)$

(c) check whether w true under the assignment $\mapsto O(p(|w|))$

(d) if false: inf.loop; otherwise halt.

"guess (satisfying) assignment $\mathcal{A}$; check in polynomial time that formula true under $\mathcal{A}$"

Cook's theorem

Theorem $SAT = \{w \mid w$ is a satisfiable formula of propositional logic$\}$ is NP-complete.

Proof (Idea) (2) We show that for all $L \in NP$, $L \preceq_{\text{pol}} SAT$

- We show that we can simulate the way a NTM works using propositional logic.
- Let $L \in NP$. There exists a p-time bounded NTM which accepts L. (Assume w.l.o.g. that M has only one tape and does not hang.)

  For M and w we define a propositional logic language and a formula $T_{M,w}$ such that

  $$M \text{ accepts } w \quad \text{iff} \quad T_{M,w} \text{ is satisfiable.}$$
- We show that the map f with $f(w) = T_{M,w}$ has polynomial complexity.

Closure of complexity classes

P, PSPACE are closed under complement

All complexity classes which are defined in terms of deterministic Turing machines are closed under complement.

Proof: If a language L is in such a class then also its complement is

(run the machine for L and revert the output)

Closure of complexity classes

Is NP closed under complement?

Closure of complexity classes

Is NP closed under complement?

Nobody knows!

Definition

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

$$\text{co-NP} = \{L \mid \overline{L} \in NP\}$$

Relationships between complexity classes

It is not yet known whether the following relationships hold:

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

$\mathsf{NP} \stackrel{?}{=} \text{co-}\mathsf{NP}$

$\mathsf{P} \stackrel{?}{=} \mathsf{PSPACE}$

$\mathsf{NP} \stackrel{?}{=} \mathsf{PSPACE}$

Examples of NP-complete problems

Examples of NP-complete problems:

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

Examples of NP-complete problems

Examples of NP-complete problems:

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

Examples of NP-complete problems

Definition (CNF, DNF, *k*-CNF, *k*-DNF)

DNF: A formula is in DNF if it has the form

$$(L_1^1 \wedge \cdots \wedge L_{n_1}^1) \vee \cdots \vee (L_1^m \wedge \cdots \wedge L_{n_m}^m)$$

CNF: A formula is in CNF if it has the form

$$(L_1^1 \vee \cdots \vee L_{n_1}^1) \wedge \cdots \wedge (L_1^m \vee \cdots \vee L_{n_m}^m)$$

k-DNF: A formula is in k-DNF if it is in DNF and all its conjunctions have k literals

k-CNF: A formula is in k-CNF if it is in CNF and all its disjunctions have k literals

Examples of NP-complete problems

$\text{SAT} = \{w \mid w \text{ is a satisfiable formula of propositional logic}\}$

$\text{CNF-SAT} = \{w \mid w \text{ is a satisfiable formula of propositional logic in CNF}\}$

$k\text{-CNF-SAT} = \{w \mid w \text{ is a satisfiable formula of propositional logic in } k\text{-CNF}\}$

Examples of NP-complete problems

Theorem

The following problems are in NP and are NP-complete:

(1) SAT

(2) CNF-SAT

(3) k-CNF-SAT for $k \geq 3$

Examples of NP-complete problems

Theorem

The following problems are in NP and are NP-complete:

(1) SAT

(2) CNF-SAT

(3) k-CNF-SAT for $k \geq 3$

Proof: (1) SAT is NP-complete by Cook's theorem.

CNF-SAT and k-CNF-SAT are clearly in NP.

(3) We show that 3-CNF-SAT is NP-hard. For this, we construct a polynomial reduction of SAT to 3-CNF-SAT.

Examples of NP-complete problems

Proof: (ctd.) Polynomial reduction of SAT to 3-CNF.

Let F be a propositional formula of length n

Step 1 Move negation inwards (compute the negation normal form) $\mapsto O(n)$

Step 2 Fully bracket the formula $\mapsto O(n)$

$P \wedge Q \wedge R \mapsto (P \wedge Q) \wedge R$

Step 3 Starting from inside out replace subformula $Q \circ R$ with a new propositional variable $P_{Q \circ R}$ and add the formula

$P_{Q \circ R} \rightarrow (Q \circ R)$ and $(Q \circ R) \rightarrow P_{Q \circ R}$ $(\circ \in \{\vee, \wedge\})$ $\mapsto O(p(n))$

Step 4 Write all formulae above as clauses $\mapsto$ Rename(F) $\mapsto O(n)$

Let $f : \Sigma^* \rightarrow \Sigma^*$ be defined by:

$f(F) = P_F \wedge$ Rename(F) if F is a well-formed formula

and $f(w) = \perp$ otherwise. Then:

$F \in$ SAT iff F is a satisfiable formula in prop. logic iff $P_F \wedge Rename(F)$ is satisfiable

iff $f(F) \in$ 3-CNF-SAT

Example

Let F be the following formula:

$$[(Q \wedge \neg P \wedge \neg(\neg(\neg Q \vee \neg R))) \vee (Q \wedge \neg P \wedge \neg(Q \wedge \neg P))] \wedge (P \vee R).$$

Step 1: After moving negations inwards we obtain the formula:

$$F_1 = [(Q \wedge \neg P \wedge (\neg Q \vee \neg R)) \vee (Q \wedge \neg P \wedge (\neg Q \vee P))] \wedge (P \vee R)$$

Step 2: After fully bracketing the formula we obtain:

$$F_2 = [((Q \wedge \neg P) \wedge (\neg Q \vee \neg R)) \vee ((Q \wedge \neg P) \wedge (\neg Q \vee P)] \wedge (P \vee R)$$

Step 3: Replace subformulae with new propositional variables (starting inside).

$$\underbrace{\underbrace{[\underbrace{(\underbrace{(Q \wedge \neg P)}_{P_1} \wedge \underbrace{(\neg Q \vee \neg R)}_{P_2})}_{P_6} \vee \underbrace{(\underbrace{(Q \wedge \neg P)}_{P_1} \wedge \underbrace{(\neg Q \vee P)}_{P_4})}_{P_7}]}_{P_8} \wedge \underbrace{(P \vee R)}_{P_5}}_{P_F}.$$

Example

Step 3: Replace subformulae with new propositional variables (starting inside).

$$\underbrace{[\underbrace{(\underbrace{(Q \land \neg P)}_{P_1} \land \underbrace{(\neg Q \lor \neg R)}_{P_2})}_{P_6} \lor \underbrace{(\underbrace{(Q \land \neg P)}_{P_1} \land \underbrace{(\neg Q \lor P)}_{P_4})}_{P_7}]}_{P_8} \land \underbrace{(P \lor R)}_{P_5}}_{P_F} .$$

F is satisfiable iff the following formula is satisfiable:

$$\begin{array}{lllll} P_F & \land & (P_F \leftrightarrow (P_8 \land P_5) & \land & (P_1 \leftrightarrow (Q \land \neg P)) \\ & \land & (P_8 \leftrightarrow (P_6 \lor P_7)) & \land & (P_2 \leftrightarrow (\neg Q \lor \neg R)) \\ & \land & (P_6 \leftrightarrow (P_1 \land P_2)) & \land & (P_4 \leftrightarrow (\neg Q \lor P)) \\ & \land & (P_7 \leftrightarrow (P_1 \land P_4)) & \land & (P_5 \leftrightarrow (P \lor R)) \end{array}$$

can further exploit polarity

Example

Step 3: Replace subformulae with new propositional variables (starting inside).

$$\underbrace{\underbrace{[\underbrace{(\underbrace{(Q \wedge \neg P)}_{P_1} \wedge \underbrace{(\neg Q \vee \neg R)}_{P_2})}_{P_6} \vee \underbrace{(\underbrace{(Q \wedge \neg P)}_{P_1} \wedge \underbrace{(\neg Q \vee P)}_{P_4})}_{P_7}]}_{P_8} \wedge \underbrace{(P \vee R)}_{P_5}}_{P_F}.$$

F is satisfiable iff the following formula is satisfiable:

$$\begin{array}{lllll} P_F & \wedge & (P_F \rightarrow (P_8 \wedge P_5)) & \wedge & (P_1 \rightarrow (Q \wedge \neg P)) \\ & \wedge & (P_8 \rightarrow (P_6 \vee P_7)) & \wedge & (P_2 \rightarrow (\neg Q \vee \neg R)) \\ & \wedge & (P_6 \rightarrow (P_1 \wedge P_2)) & \wedge & (P_4 \rightarrow (\neg Q \vee P)) \\ & \wedge & (P_7 \rightarrow (P_1 \wedge P_4)) & \wedge & (P_5 \rightarrow (P \vee R)) \end{array}$$

Example

F is satisfiable iff the following formula is satisfiable:

$$\begin{array}{llllll}
P_F & \wedge & (P_F \rightarrow (P_8 \wedge P_5) & \wedge & (P_1 \rightarrow (Q \wedge \neg P)) \\
 & \wedge & (P_8 \rightarrow (P_6 \vee P_7)) & \wedge & (P_2 \rightarrow (\neg Q \vee \neg R)) \\
 & \wedge & (P_6 \rightarrow (P_1 \wedge P_2)) & \wedge & (P_4 \rightarrow (\neg Q \vee P)) \\
 & \wedge & (P_7 \rightarrow (P_1 \wedge P_4)) & \wedge & (P_5 \rightarrow (P \vee R))
\end{array}$$

Step 4: Compute the CNF (at most 3 literals per clause)

$$\begin{array}{llllll}
P_F & \wedge & (\neg P_F \vee P_8) \wedge (\neg P_F \vee P_5) & \wedge & (\neg P_1 \vee Q) \wedge (\neg P_1 \vee \neg P) \\
 & \wedge & (\neg P_8 \vee P_6 \vee P_7) & \wedge & (\neg P_2 \vee \neg Q \vee \neg R) \\
 & \wedge & (\neg P_6 \vee P_1) \wedge (\neg P_6 \vee P_2) & \wedge & (\neg P_4 \vee \neg Q \vee P) \\
 & \wedge & (\neg P_7 \vee P_1) \wedge (\neg P_7 \vee P_4) & \wedge & (\neg P_5 \vee P \vee R)
\end{array}$$

Examples of NP-complete problems

Proof: (ctd.) It immediately follows that CNF and k-CNF are NP-complete

Polynomial reduction from 3-CNF-SAT to CNF-SAT:

$f(F) = F$ for every formula in 3-CNF and $\perp$ otherwise.

$F \in$ 3-CNF-SAT iff $f(F) = F \in$ CNF-SAT.

Polynomial reduction from 3-CNF-SAT to k-CNF-SAT, $k > 3$

For every formula in 3-CNF:

$f(F) = F'$ (where F' is obtained from F by replacing a literal L with $\underbrace{L \vee \dots \vee L}_{k-2 \text{ times}}$).

$f(w) = \perp$ otherwise.

$F \in$ 3-CNF-SAT iff $f(F) = F' \in k$-CNF-SAT (because $F' \equiv F$)

Examples of problems in P

Theorem

The following problems are in P:

(1) DNF

(2) k-DNF for all k

(3) 2-CNF

(1) Let $F = (L_1^1 \wedge \dots \wedge L_{n_1}^1) \vee \dots \vee (L_1^m \wedge \dots \wedge L_{n_m}^m)$ be a formula in DNF.

F is satisfiable iff for some i: $(L_1^i \wedge \dots \wedge L_{n_1}^i)$ is satisfiable. A conjunction of literals is satisfiable iff it does not contain complementary literals.

(2) follows from (1)

(3) Finite set of 2-CNF formulae over a finite set of propositional variables. Resolution $\mapsto$ at most quadratically many inferences needed.

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. Is a (un)directed graph hamiltonian?
5. Can a graph be colored with three colors?
6. Multiprocessor scheduling

Examples of NP-complete problems

Definition

A clique in a graph G is a complete subgraph of G.

Clique $= \{(G, k) \mid G$ is an undirected graph which has a clique of size $k\}$

Examples of NP-complete problems

Theorem Clique is NP-complete.

Proof: (1) We show that Clique is in NP:

We can construct for instance an NTM which accepts Clique.

- M builds a set V' of nodes (subset of the nodes of G) by choosing k nodes of G (we say that M "guesses" V').
- M checks for all nodes in V' if there are nodes to all other nodes. (this can be done in polynomial time)

"guess a subgraph with k vertices; check in polynomial time that it is a clique"

Examples of NP-complete problems

Theorem Clique is NP-complete.

Proof: (2) We show that Clique is NP-hard by showing that 3-CNF-SAT $\preceq_{\text{pol}}$ Clique.

Let $\mathcal{G}$ be the set of all undirected graphs. We want to construct a map f (DTM computable in polynomial time) which associates with every formula F in 3-CNF a pair $f(F) = (G_F, k_F) \in \mathcal{G} \times \mathbb{N}$ such that

$$F \in \text{3-CNF-SAT} \quad \text{iff} \quad G_F \text{ has a clique of size } k_F.$$

$$F \in \text{3-CNF} \Rightarrow F = (L_1^1 \vee L_2^1 \vee L_3^1) \wedge \dots \wedge (L_1^m \vee L_2^m \vee L_3^m)$$

F satisfiable iff there exists an assignment $\mathcal{A}$ such that in every clause in F at least one literal is true and it is impossible that P and $\neg P$ are true at the same time.

Examples of NP-complete problems

Theorem Clique is NP-complete.

Proof: (ctd.) Let $k_F := m$ (the number of clauses). We construct G_F as follows:

- **Vertices:** all literals in F.
- **Edges:** We have an edge between two literals if they (i) can become true in the same assignment and (ii) belong to different clauses.

Then:
(1) $f(F)$ is computable in polynomial time.
(2) The following are equivalent:

(a) G_F has a clique of size k_F.

(b) There exists a set of nodes $\{L^1_{i_1}, \dots, L^m_{i_m}\}$ in G_F which does not contain complementary literals.

(c) There exists an assignment which makes F true.

(d) F is satisfiable.

Examples of NP-complete problems

Examples of NP-complete problems:

1. Is a logical formula satisfiable? (SAT, 3-CNF-SAT)
2. Does a graph contain a clique of size k?
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
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.

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

Examples of NP-complete problems

Theorem Rucksack is NP-complete.

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

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 $\{x_1, x_2, x_3, x_4, x_5\}$.

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

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

Satisfying assignment: $\mathcal{A}(x_1)=\mathcal{A}(x_2)=\mathcal{A}(x_5)=1$ 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

$\mathcal{A}$ 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

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

COLOR $= \{(G, k) \mid G$ undirected graph that can be colored with k colors$\}$

COLOR 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

COLOR $= \{(G, k) \mid G$ undirected graph that can be colored with k colors$\}$

COLOR is NP-complete

Detailed proof: Available online from the website

(file: k-coloring-np-complete-proof.pdf)

3-colorability $= \{G \mid G$ undirected graph that can be colored with 3 colors$\}$

3-colorability is NP-complete

(for a proof see e.g. https://cgi.csc.liv.ac.uk/ igor/COMP309/3CP.pdf)

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-path)

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 and is a cycle.

$L_{\text{Ham,undir}}$: 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 a 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.

It is not known whether NP = co-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:
$(\forall P)F \mapsto F[P \mapsto 1] \land F[P \mapsto 0]$
$(\exists P)F \mapsto F[P \mapsto 1] \lor F[P \mapsto 0]$

PSPACE

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

Quantified Boolean Formulas (QBF)

Given: A well-formed quantified Boolean formula

$$F = (Q_1 P_1) \dots (Q_n P_n) G(P_1, \dots, P_n)$$

where G is a Boolean expression containing the propositional variables $P_1, \dots, P_n$ and Q_i is $\exists$ or $\forall$.

Question: Is F true?

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

PSPACE

Example

F propositional formula with propositional variables $P_1, \ldots, P_n$

F is satisfiable iff $\exists P_1 \ldots \exists P_n F$ is true.

PSPACE

Example

F propositional formula with propositional variables $P_1, \dots, P_n$

F is satisfiable iff $\exists P_1 \dots \exists P_n F$ is true.

If we have alternations of quantifiers it is more difficult to check whether a QBF is true.

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

NP vs. Co-NP

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

Example:

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

NP vs. Co-NP

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

Example:

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

Informally

$L \in NP$ iff there exists a language $L' \in P$ and a $k \geq 0$ s.t. for all $w \in \Sigma^*$:

$w \in L$ iff $\exists$ c (witness) of lenght polynomial in $|w|$ and s.t. $\langle w, c \rangle \in L'$
(can use c to check in PTIME that $w \in L$)

NP vs. Co-NP

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

Example:

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

Informally

$L \in$ NP iff there exists a language $L' \in$ P and a $k \geq 0$ s.t. for all $w \in \Sigma^*$:

$w \in L$ iff $\exists$ c (witness) of lenght polynomial in $|w|$ and s.t. $\langle w, c \rangle \in L'$
(can use c to check in PTIME that $w \in L$)

$L \in$ co-NP iff the complement of L is in NP (with test language L')

$w \in L$ iff $\forall c$ of lenght polynomial in $|w|$, $\langle w, c \rangle \notin L'$
(can use c to check in PTIME that $w \in L$)

NP vs. Co-NP

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

Example:

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

Informally

$L \in$ NP iff there exists a PTIME deterministic verifyer M s.t. for all $w \in \Sigma^*$:

$w \in L$ iff $\exists$ c (witness) of lenght polynomial in $|w|$ and s.t. $M(w, c) = 1$

$L \in$ co-NP iff the complement of L is in NP (with test language L')

$w \in L$ iff $\forall c$ of lenght polynomial in $|w|$, $M(w, c) = 1$.

The structure of PSPACE

... Beyond NP

The structure of PSPACE

Idea: (M PTIME deterministic verifyer)

NP

$w \in L$ iff $\exists\ c$ (witness) of lenght polynomial in $|w|$ s.t. $M(w, c) = 1$.

co-NP

$w \in L$ iff $\forall c$ of lenght polynomial in $|w|$, s.t. $M(w, c) = 1$.

Σ_2^p

$w \in L$ iff $\exists\ c$ (witness) of lenght polynomial in $|w|$ s.t.
$\forall d$ of lenght polynomial in $|w|$, $M(w, c, d) = 1$

The structure of PSPACE

Idea: (M PTIME deterministic verifyer)

NP

$w \in L$ iff $\exists$ c (witness) of lenght polynomial in $|w|$ s.t. $M(w, c) = 1$.

co-NP

$w \in L$ iff $\forall c$ of lenght polynomial in $|w|$, s.t. $M(w, c) = 1$.

Σ_2^p

$w \in L$ iff $\exists$ c (witness) of lenght polynomial in $|w|$ s.t.
$\forall d$ of lenght polynomial in $|w|$, $M(w, c, d) = 1$

Example: QBF with one quantifier alternation

$\Sigma_2 SAT = \{F = \exists P_1 \dots P_n \forall Q_1 \dots Q_m \overline{F}(P_1, \dots, P_n, Q_1, \dots Q_n) \mid F \text{ true}\}$

The structure of PSPACE

Remarks

- in fact, $\Sigma_2 SAT$ is complete for Σ_2^p
- more alternations lead to a whole hierarchy
- all of it is contained in PSPACE

The structure of PSPACE

For $i \geq 1$,a language L is in Σ_i^p if there exists a PTIME deterministic verifyer M such that:

$$\begin{array}{lll} w \in L & \text{iff} & \exists u_1 \text{ of lenght polynomial in } |w| \\ & & \forall u_2 \text{ of lenght polynomial in } |w| \\ & & \ldots \\ & & Q_i u_i \text{ of lenght polynomial in } |w| \\ & & \text{such that } M(w, u_1, \ldots, u_i) = 1 \end{array}$$

where Q_i is $\exists$ if i is odd and $\forall$ otherwise.

The polynomial hierarchy is the set $PH = \bigcup_{i \geq 1} \Sigma_i^p$

$\Pi_i^p = \text{co-}\Sigma_i^p = \{\overline{L} \mid L \in \Sigma_i^p\}$

The structure of PSPACE

Formal definition (main ideas)
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.

The structure of PSPACE

The polynomial hierarchy (Informally)

P^Y : the class of languages decidable in polynomial time by a Turing machine augmented by an oracle for some complete problem in class Y.

NP^Y : the class of languages decidable in polynomial time by a non-deterministic Turing machine augmented by an oracle for some complete problem in class Y.

A^B : the class of languages decidable by an algorithm in class A with an oracle for some complete problem in class B.

The structure of PSPACE

The polynomial hierarchy (Informally)

A^B : the class of languages decidable by an algorithm in class A with an oracle for some complete problem in class B.

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

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

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

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

The structure of PSPACE

$$\begin{tikzcd}
 & \mathrm{PSPACE} & \\
 & \vdots & \\
\Sigma_3^{\mathrm{P}} & & \Pi_3^{\mathrm{P}} \\
 & \Delta_3^{\mathrm{P}} \arrow[ul] \arrow[ur] & \\
\Sigma_2^{\mathrm{P}} \arrow[uu] \arrow[ur] & & \Pi_2^{\mathrm{P}} \arrow[uu] \arrow[ul] \\
 & \mathrm{P}^{\mathrm{NP}} = \Delta_2^{\mathrm{P}} \arrow[ul] \arrow[ur] & \\
\mathrm{NP} = \Sigma_1^{\mathrm{P}} \arrow[uu] \arrow[ur] & & \Pi_1^{\mathrm{P}} = \mathrm{coNP} \arrow[uu] \arrow[ul] \\
 & \Delta_0^{\mathrm{P}} = \Sigma_0^{\mathrm{P}} = \mathrm{P} = \Pi_0^{\mathrm{P}} = \Delta_1^{\mathrm{P}} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

The structure of PSPACE

It is an open problem whether there is an i such that $\Sigma_i^p = \Sigma_{i+1}^p$.

This would imply that $\Sigma_i^p = PH$: the hierarchy collapses to the i-th level.

Most researchers believe that the hierarchy does not collapse.

If NP $=$ P then PH $=$ P, i.e. the hierarchy collapses to P.

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

Parameterized 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:

- Identify which part of the input is cause of high complexity
- 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.

(I) Parameterized Complexity

Parameterized complexity is a branch of computational complexity theory that focuses on classifying computational problems according to their inherent difficulty with respect to multiple parameters of the input.

This allows the classification of NP-hard problems on a finer scale.

$\mapsto$ Fixed parameter tractability.

Example: SAT
Assume that the number of propositional variables is a parameter.

A given formula of size m with k variables can be checked by brute force in time $O(2^k m)$

For a fixed number of variables, the complexity of the problem is linear in the length of the input formula.

(I) Parameterized Complexity

Fixed parameter tractability parameter specified: Input of the form (w, k)
L is fixed-parameter tractable if the question $(w, k) \in L$? can can be decided in running time $f(k) \cdot p(|w|)$, where f is an arbitrary function depending only on k, and p is a polynomial.

An example of a problem that is thought not to be fixed parameter tractable is graph coloring parameterised by the number of colors.

It is known that 3-coloring is NP-hard, and an algorithm for graph k-colouring in time $f(k)p(n)$ for $k = 3$ would run in polynomial time in the size of the input.

Thus, if graph coloring parameterised by the number of colors were fixed parameter tractable, then P = NP.

(II) 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 P = NP)

(III) 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