Computability and Complexity

Lecture 10

Polynomial time verifiers
More problems in NP
Polynomial time reducibility

given by Jiri Srba

Motivation Example: *HAMPATH*

$$HAMPATH \stackrel{\text{def}}{=} \{\langle G, s, t\rangle \mid G \text{ is a digraph with a Hamiltonian path from } s \text{ to } t\}$$

Last time we showed that $HAMPATH \in \mathrm{NP}$.

Motivation Example: *HAMPATH*

$$\mathit{HAMPATH} \stackrel{\text{def}}{=} \{\langle G, s, t\rangle \mid G \text{ is a digraph with a Hamiltonian path from } s \text{ to } t\ \}$$

Last time we showed that $\mathit{HAMPATH} \in \mathrm{NP}$.

Another view:

- Assume that some solution, called certificate (in our case a path in G), is given to us.
- In polynomial time we can verify whether it is a Hamiltonian path from s to t or not.

Motivation Example: *HAMPATH*

$$\mathit{HAMPATH} \stackrel{\text{def}}{=} \{\langle G, s, t\rangle \mid$$
$$G \text{ is a digraph with a Hamiltonian path from } s \text{ to } t\ \}$$

Last time we showed that $\mathit{HAMPATH} \in \mathrm{NP}$.

Another view:

- Assume that some solution, called certificate (in our case a path in G), is given to us.
- In polynomial time we can verify whether it is a Hamiltonian path from s to t or not.

- Hence *HAMPATH* has a polynomial time verifiable.

Polynomial Time Verifiable Languages

Definition

- A verifier for a language L is a decider V such that
$$L = \{w \mid \ V \text{ accepts } \langle w, c\rangle \text{ for some string } c\}$$
- The string c is called a certificate or a proof.
- A verifier is called polynomial time verifier if it runs in a polynomial time in the length of w (hence the length of c is irrelevant).
- A language L is polynomial time verifiable language if it has a polynomial time verifier.

Polynomial Time Verifiable Languages

Definition

- A verifier for a language L is a decider V such that

$$L = \{w \mid \; V \text{ accepts } \langle w, c\rangle \text{ for some string } c\}$$

- The string c is called a certificate or a proof.
- A verifier is called polynomial time verifier if it runs in a polynomial time in the length of w (hence the length of c is irrelevant).
- A language L is polynomial time verifiable language if it has a polynomial time verifier.

Example:

- $HAMPATH$ is a polynomial time verifiable language.
- We do not know whether $\overline{HAMPATH}$ has a polynomial time verifier or not.

Poly-Time Verifiers vs. Nondeterministic Poly-Time TMs

Theorem

A language L is polynomial time verifiable if and only if L is decidable in polynomial time by a nondeterministic TM.

Poly-Time Verifiers vs. Nondeterministic Poly-Time TMs

Theorem

A language L is polynomial time verifiable if and only if L is decidable in polynomial time by a nondeterministic TM.

"$\Rightarrow$": Let V be a verifier for L running in time $O(n^k)$. We construct a nondeterministic decider M for L:

$M =$ "On input w of length n:

1. Nondeterministically select a string c of length $\leq k_1 n^k$.
2. Run V on $\langle w, c \rangle$. Accept if and only if V accepted."

Poly-Time Verifiers vs. Nondeterministic Poly-Time TMs

Theorem

A language L is polynomial time verifiable if and only if L is decidable in polynomial time by a nondeterministic TM.

"$\Rightarrow$": Let V be a verifier for L running in time $O(n^k)$. We construct a nondeterministic decider M for L:

$M =$ "On input w of length n:

1. Nondeterministically select a string c of length $\leq k_1 n^k$.
2. Run V on $\langle w, c \rangle$. Accept if and only if V accepted."

"$\Leftarrow$": Let M be a nondeterministic polynomial time decider for L. We construct a polynomial time verifier V for L:

$V =$ "On input $\langle w, c \rangle$ where w and c are strings:

1. Simulate one particular branch of M run on w where nondeterministic choices are given by c.
2. Accept if and only if this branch accepted."

Further Languages in NP and the Class co-NP

$CLIQUE \stackrel{\text{def}}{=} \{\langle G, k\rangle \mid\ G \text{ is a graph with a } k\text{-clique } \}$

Theorem

CLIQUE is in NP.

$CLIQUE \stackrel{\text{def}}{=} \{\langle G, k\rangle \mid G \text{ is a graph with a } k\text{-clique } \}$

Theorem

CLIQUE is in NP.

$SUBSET\text{-}SUM \stackrel{\text{def}}{=} \{\langle S, t\rangle \mid$
$S = \{x_1, \ldots, x_k\} \subseteq \mathbb{N},\ t \in \mathbb{N}, \text{ and there is } X \subseteq S \text{ s.t. } \sum X = t\ \}$

Theorem

SUBSET-SUM is in NP.

Further Languages in NP and the Class co-NP

$CLIQUE \stackrel{\text{def}}{=} \{\langle G, k\rangle \mid \; G \text{ is a graph with a } k\text{-clique } \}$

Theorem

CLIQUE is in NP.

$SUBSET\text{-}SUM \stackrel{\text{def}}{=} \{\langle S, t\rangle \mid$
$S = \{x_1, \ldots, x_k\} \subseteq \mathbb{N},\ t \in \mathbb{N}, \text{ and there is } X \subseteq S \text{ s.t. } \sum X = t \ \}$

Theorem

SUBSET-SUM is in NP.

Observe, that $\overline{CLIQUE}$ and $\overline{SUBSET\text{-}SUM}$ are not necessarily in NP (in fact, we do not know if they are or not).

Definition (The class co-NP)

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

Summary of Time Complexity Classes

Complexity Class P

- contains languages decidable in deterministic polynomial time

Complexity Class NP

- contains languages decidable in nondeterm. polynomial time
- contains languages that have polynomial time verifiers

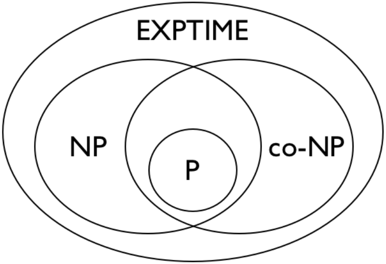
Complexity Class co-NP

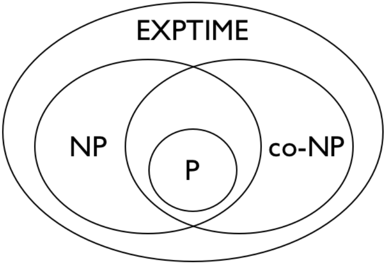
- contains complements of all languages that are in NP

Complexity Class EXPTIME

- contains languages decidable in deterministic exponential time
- $\mathsf{EXPTIME} \stackrel{\text{def}}{=} \bigcup_{k \geq 0} \mathsf{TIME}(2^{n^k})$

Time Complexity Classes





Remarks:

- $P \subseteq NP \subseteq EXPTIME$, and $P \subseteq \text{co-}NP \subseteq EXPTIME$
- We know that $P \neq EXPTIME$, but
- the strictness of the other inclusions, as well as the question whether $NP = \text{co-}NP$, are still open!

Definition (Polynomial Time Computable Function)

A function $f : \Sigma^* \to \Sigma^*$ is a polynomial time computable function iff there exists a TM M_f running in polynomial time which on any given input $w \in \Sigma^*$

- always halts, and
- leaves just $f(w)$ on its tape.

Polynomial Time Reducibility

Definition (Polynomial Time Computable Function)

A function $f : \Sigma^* \rightarrow \Sigma^*$ is a polynomial time computable function iff there exists a TM M_f running in polynomial time which on any given input $w \in \Sigma^*$

- always halts, and
- leaves just $f(w)$ on its tape.

Definition (Polynomial Time Mapping Reducibility, $A \leq_P B$)

Let $A, B \subseteq \Sigma^*$. We say that language A is polynomial time (mapping) reducible to language B, written $A \leq_P B$, iff

1. there is a polynomial time computable function $f : \Sigma^* \rightarrow \Sigma^*$ such that
2. for every $w \in \Sigma^*$: $w \in A$ if and only if $f(w) \in B$

Theorem

Theorem

If $A \leq_P B$ and $B \in \mathrm{P}$, then $A \in \mathrm{P}$.

Theorem

If $A \leq_P B$ and $B \in \mathrm{P}$, then $A \in \mathrm{P}$.

Proof:

- Let f be a polynomial time reduction from A to B computed by a machine M_f running in time $O(n^k)$.
- Let M_B be a decider for B running in time $O(n^\ell)$.

We construct a polynomial time decider M_A for A:

$M_A =$ "On input w:
1. Run M_f on w (it halts and leaves $f(w)$ on the tape).
2. Run M_B on $f(w)$ and accept iff M_B accepted."

Theorem

If $A \leq_P B$ and $B \in \mathrm{P}$, then $A \in \mathrm{P}$.

Proof:

- Let f be a polynomial time reduction from A to B computed by a machine M_f running in time $O(n^k)$.
- Let M_B be a decider for B running in time $O(n^\ell)$.

We construct a polynomial time decider M_A for A:

$M_A =$ "On input w:
1. Run M_f on w (it halts and leaves $f(w)$ on the tape).
2. Run M_B on $f(w)$ and <u>accept</u> iff M_B accepted."

Step 1. runs in time $O(n^k)$ and outputs $f(w)$ of length $O(n^k)$.
Step 2. runs in time $O((n^k)^\ell) = O(n^{k \cdot \ell})$. $\square$

SAT Problem

Let $V = \{x_1, x_2, \ldots, y, z, \ldots\}$ be a set of Boolean variables.

Definition (Boolean Formulae)

The set of Boolean formulae is defined by the abstract syntax:

$$\phi ::= x \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi$$

where x ranges of the set of variables.

A Boolean formula ϕ is satisfiable if there is an assignment of truth values to the variables on which ϕ evaluates to true.

SAT Problem

Let $V = \{x_1, x_2, \ldots, y, z, \ldots\}$ be a set of Boolean variables.

Definition (Boolean Formulae)

The set of Boolean formulae is defined by the abstract syntax:

$$\phi ::= x \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi$$

where x ranges of the set of variables.

A Boolean formula ϕ is satisfiable if there is an assignment of truth values to the variables on which ϕ evaluates to true.

Definition (The language SAT)

$$SAT \stackrel{\text{def}}{=} \{\langle\phi\rangle \mid \ \phi \text{ is a satisfiable Boolean formula } \}$$

3SAT Problem

- Literal is a variable or a negation of a variable.
- Instead of $\neg x$ we usually write $\overline{x}$.
- Clause of size 3 is a disjunction of three literals.
- A formula in 3-cnf (3 conjunctive normal form) is a conjunction of clauses of size 3.

Example of a 3-cnf formula with 4 clauses:

$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_3 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3} \vee x_3)$$

- Literal is a variable or a negation of a variable.
- Instead of $\neg x$ we usually write $\overline{x}$.
- Clause of size 3 is a disjunction of three literals.
- A formula in 3-cnf (3 conjunctive normal form) is a conjunction of clauses of size 3.

Example of a 3-cnf formula with 4 clauses:

$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_3 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3} \vee x_3)$$

Definition (The language *3SAT*)

$$\mathit{3SAT} \stackrel{\text{def}}{=} \{\langle \phi \rangle \mid \ \phi \text{ is a satisfiable 3-cnf formula } \}$$

Polynomial Time Reduction from 3SAT to $CLIQUE$

Theorem

$3SAT \leq_P CLIQUE$

Theorem

$3SAT \leq_P CLIQUE$

Proof: For a $3SAT$ instance with k clauses

$$\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_k \vee b_k \vee c_k)$$

construct in poly-time an instance $G = (V, E), k$ of $CLIQUE$ s.t.

formula ϕ is satisfiable if and only if G has a k-clique.

Theorem

$3SAT \leq_P CLIQUE$

Proof: For a $3SAT$ instance with k clauses

$$\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_k \vee b_k \vee c_k)$$

construct in poly-time an instance $G = (V, E), k$ of $CLIQUE$ s.t.

formula ϕ is satisfiable if and only if G has a k-clique.

V ... every occurence of a literal in ϕ is a node ($3k$ nodes)
E ... all possible connections, without

- edges between literals in the same clause, and without
- edges between contradictory literals (x and $\overline{x}$).

□

Exam Questions

- Polynomial time verifiers and their equivalence with nondeterministic polynomial time TMs.
- *CLIQUE*, *SUBSET-SUM* are in NP.
- Classes co-NP and EXPTIME.
- Polynomial time reducibility and $3SAT \leq_P CLIQUE$.