Fundamental Computer Science
Lecture 4: Complexity
NP-completeness

Denis Trystram
MoSIG1 and M1Info – University Grenoble-Alpes

March, 2021

Last lecture

- Definition of time complexity classes
  - $\mathcal{P}$: problems solvable in $O(n^k)$ time
  - $\mathcal{NP}$: problems verifiable in $O(n^k)$ time
  - space complexity

- Prove that a problem belongs to $\mathcal{NP}$
  - provide a polynomial-time *verifier*

- Reduction from problem A to problem B $\quad (A \leq_{\mathrm{P}} B)$
  1. transform an instance I_{A} of A to an instance I_{B} of B
  2. show that the reduction is of polynomial size
  3. prove that:

     "there is a solution for the problem A on the instance I_{A}

     if and only if

     there is a solution for the problem B on the instance I_{B}"

Agenda

- Definition of the class NP-complete
- The SAT problem
- Cook-Levin theorem
- Use reductions to prove NP-completeness
- A detailed example: Vertex Cover
- Variants of SAT

Completeness

Let $\mathcal{C}$ be a set of languages.

Definition

A language B is $\mathcal{C}$-complete if

- $B \in \mathcal{C}$, and
- every language A in $\mathcal{C}$ is polynomially reducible to B.

NP-COMPLETENESS

Definition

A language B is NP-COMPLETE if

- $B \in \mathcal{NP}$, and
- every language A in $\mathcal{NP}$ is polynomially reducible to B.

NP-completeness

Definition

A language B is NP-complete if

- $B \in \mathcal{NP}$, and
- every language A in $\mathcal{NP}$ is polynomially reducible to B.

Theorem

If B is NP-complete *and $B \in \mathcal{P}$, then $\mathcal{P} = \mathcal{NP}$*

Proof:

- direct from the definition of reducibility

NP-COMPLETENESS

Definition

A language B is NP-COMPLETE if

- $B \in \mathcal{NP}$, and
- every language $A \in \mathcal{NP}$ is polynomially reducible to B.

Theorem

If B is NP-COMPLETE *and $B \leq_P C$ for $C \in \mathcal{NP}$, then C is* NP-COMPLETE

Proof:

- initially, $C \in \mathcal{NP}$
- we need to show: **every $A \in \mathcal{NP}$ polynomially reduces to C**
  - every language $\in \mathcal{NP}$ polynomially reduces to B
  - B polynomially reduces to C

The next step

Prove that there are some problems in $\text{NP-}\textsc{complete}$

Stephen Cook proved in 1971 that $SAT \in \text{NP-}\textsc{complete}$

Recall on Logic: Boolean formulas

- x_i: a Boolean variable, values TRUE or FALSE
- $\bar{x}_i$: negation of x_i – logical NOT
- $x_i, \bar{x}_i$: literals

Recall on Logic: Boolean formulas

- x_i: a Boolean variable, values TRUE or FALSE
- $\bar{x}_i$: negation of x_i – logical NOT
- $x_i, \bar{x}_i$: literals
- $\vee$: logical OR
- $\wedge$: logical AND

Recall on Logic: Boolean formulas

- x_i: a Boolean variable, values TRUE or FALSE
- $\bar{x}_i$: negation of x_i – logical NOT
- $x_i, \bar{x}_i$: literals
- $\vee$: logical OR
- $\wedge$: logical AND
- $(x_1 \vee \bar{x}_3 \vee x_4)$: clause, a set of literals in disjunction

Recall on Logic: Boolean formulas

- x_i: a Boolean variable, values TRUE or FALSE
- $\bar{x}_i$: negation of x_i – logical NOT
- $x_i, \bar{x}_i$: literals
- $\vee$: logical OR
- $\wedge$: logical AND
- $(x_1 \vee \bar{x}_3 \vee x_4)$: clause, a set of literals in disjunction
- $\mathcal{F} = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_4) \wedge (x_1 \vee x_4)$:
  a Boolean formula in Conjunctive Normal Form (CNF), a set of clauses in conjunction
  - every formula can be written in CNF (thus, focus on CNF formulas)

Recall on Logic: Boolean formulas

- x_i: a Boolean variable, values TRUE or FALSE
- $\bar{x}_i$: negation of x_i – logical NOT
- $x_i, \bar{x}_i$: literals
- $\vee$: logical OR
- $\wedge$: logical AND
- $(x_1 \vee \bar{x}_3 \vee x_4)$: clause, a set of literals in disjunction
- $\mathcal{F} = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_4) \wedge (x_1 \vee x_4)$:
  a Boolean formula in Conjunctive Normal Form (CNF), a set of clauses in conjunction
  - every formula can be written in CNF (thus, focus on CNF formulas)
- *assignment*: give TRUE or FALSE value to variables

Recall on Logic: Boolean formulas

- x_i: a Boolean variable, values TRUE or FALSE
- $\bar{x}_i$: negation of x_i – logical NOT
- $x_i, \bar{x}_i$: literals
- $\vee$: logical OR
- $\wedge$: logical AND
- $(x_1 \vee \bar{x}_3 \vee x_4)$: clause, a set of literals in disjunction
- $\mathcal{F} = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_4) \wedge (x_1 \vee x_4)$:
  a Boolean formula in Conjunctive Normal Form (CNF), a set of clauses in conjunction
  - every formula can be written in CNF (thus, focus on CNF formulas)
- *assignment*: give TRUE or FALSE value to variables
- a formula is *satisfiable* if there is an assignment evaluating to TRUE
  - i.e, $(x_1, x_2, x_3, x_4) = (\mathrm{T}, \mathrm{T}, \mathrm{T}, \mathrm{F})$ for the above formula $\mathcal{F}$

The satisfiability problem SAT

- $X = \{x_1, x_2, \ldots, x_n\}$: set of variables
- $C = \{C_1, C_2, \ldots, C_m\}$: set of clauses
- $\mathcal{F} = C_1 \wedge C_2 \wedge \ldots \wedge C_m$

$$\text{SAT} = \{\langle \mathcal{F} \rangle \mid \mathcal{F} \text{ is a satisfiable Boolean formula } \}$$

The problem version of SAT:

- SAT
- **Instance.** m clauses C_i expressed using n literals
- **Question.** Is the formula $\mathcal{F} = C_1 \wedge C_2 \wedge ... \wedge C_m$ satisfiable?

Example: Vertex Cover

We will show in a separate lesson that $VC \in \mathrm{NP}\text{-}\textsc{complete}$