Complexity Theory

(class 7 of 15)

Álvaro Moreira
alvaro.moreira@inf.ufrgs.br

Instituto de Informática
Universidade Federal do Rio Grande do Sul
Porto Alegre, Brasil
http://www.inf.ufrgs.br

Relation between NP and P

- Recall that:

  Definition 1.13 (The class P) $\mathrm{P} = \bigcup_{c \geqslant 1} \mathrm{DTIME}(n^c)$.

- Now let's define
$$\mathrm{EXP} = \bigcup_{c \geqslant 1} \mathrm{DTIME}(2^{n^c})$$

- The following relation between P and NP and EXP is trivial:

  Claim 2.4 $\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{EXP}$

Relation between P and NP (II)

Proof of $\mathrm{P} \subseteq \mathrm{NP}$:

Suppose $L \in \mathrm{P}$ is decided in polynomial-time by a TM N. Then $L \in \mathrm{NP}$ since we can take N as the machine M in **Definition 2.1** and make $p(|x|)$ the zero polynomial (in other words, u is an empty string).

Proof of $\mathrm{NP} \subseteq \mathrm{EXP}$:

Suppose $L \in \mathrm{NP}$ and suppose TM M and the polynomial p are as in **Definition 2.1** . We can **decide** L, i.e. we can **decide** if a string $x \in L$, by enumerating **all** possible strings u of size at most $p(|x|)$ (i.e all $u \in \{0, 1\}^{p(|x|)}$), and by using M to check whether one of these strings is a **certificate** for the input x. If there is such a string then $x \in L$ otherwise if all strings u are tested by M and none of them is a certificate, the conclusion is that $x \notin L$.

Relation between P and NP (III)

Note that the machine M used above runs in polynomial time (see **Definition 2.1**) but it might have to check all possible strings u.

The size of each u (according to **Definition 2.1**) is at most polynomial in the size n of the input x. In other words the size of each u is $O(n^c)$ for some $c > 1$.

Since the number of strings u of size $O(n^c)$ in $\{0, 1\}^*$ is $2^{O(n^c)}$ (exponential), and since M might have to check all of them, the number of times that we might have to run M do decide L is exponential. Hence $L \in \mathrm{EXP}$ $\square$

Reduction (I)

- The independent set problem is an example of a problem at least as hard as any other language in $\mathbf{NP}$
- That means that, if independent set problem has a polynomial-time algorithm then so do all the problems in $\mathbf{NP}$.
- This property is called NP-hardness.
- How can we prove that a language B is at least as hard as some other language A? (or, equivalently, how can we prove that a problem B is at least as hard as some other problem A?)
- The crucial tool is the notion of a **polynomial reduction**

Reduction (II)

Definition 2.7 (Reductions, NP-hardness and NP-completeness)

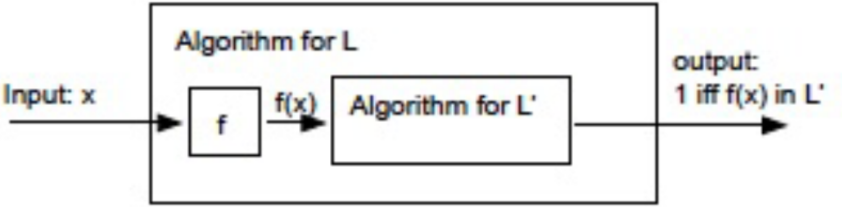
- *We say that a language $L \subseteq \{0,1\}^*$ is* ***polynomial-time Karp reducible*** *to a language $L' \subseteq \{0,1\}^*$, denoted by $L \leqslant_p L'$, if there is a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for every $x \in \{0,1\}^*$, $x \in L$ iff $f(x) \in L'$.*

- *We say that L' is* ***NP-hard*** *i $L \leqslant_p L'$ for every $L \in NP$.*

- *We say that L' is* ***NP-complete*** *if L' is NP-hard and $L' \in NP$.*

Reduction (III)

The figure below illustrates the process of reduction of a language L to a language L':



Observe that if $\mathrm{L} \leqslant_{\mathrm{p}} \mathrm{L}'$ and if $\mathrm{L}' \in \mathrm{P}$ (i.e. the algorithm for L' in the figure above is polynomial) then $\mathrm{L} \in \mathrm{P}$

Reduction and NP-Completeness

For proving the theorem below note that if p and q are two functions that grow at most n^c and n^d respectively, then the function $p \circ q$ grows at most n^{cd}

Theorem 2.8

1. (Transitivity) If $L \leqslant_p L'$ and $L' \leqslant_p L''$, then $L \leqslant_p L''$.

2. If language L is NP-hard and $L \in P$ then $P = NP$.

3. If language l is NP-complete then $L \in P$ if and only if $P = NP$.

Polynomial reduction is transitive

Below is the proof that if $L \leqslant_p L'$ and $L' \leqslant_p L''$, then $L \leqslant_p L''$:

- If f_1 is a polynomial-time reduction from L to L' and f_2 is a polynomial-time reduction from L' to L'' then the function $f_2 \circ f_1$ is a polynomial-time reduction from L to L'' since:

  (i) given x, $f_2 \circ f_1$ takes polynomial-time to compute $(f_2 \circ f_1)(x)$ (see observation before **Theorem 2.8**), and

  (ii) since $x \in L$ iff $f_1(x) \in L'$ and since $f_1(x) \in L'$ iff $f_2(f_1(x)) \in L'$, we have that $x \in L$ iff $f_2(f_1(x)) \in L'$

Exercise: Prove 2 and 3 of **Theorem 2.8**

NP-Completeness (I)

- According to **Definition 2.7** a language is NP-Complete if it is $\mathrm{NP} - \mathrm{Hard}$ and if is in NP.
- Stephen Cook and Leonid Levin independently were the first to prove that a problem is NP-Complete (the problem $\mathcal{SAT}$)
- They did that by proving that

  (i) all problems in NP can be reduced to $\mathcal{SAT}$ (i.e that $\mathcal{SAT}$ is NP-Hard), and that

  (ii) $\mathcal{SAT}$ is a problem in NP

NP-Completeness (II)

- Later Karp showed that many problems of practical interest are NP-Complete problems

- To prove that a language L is NP-Complete, Karp didn't use the approach used by Cook and Levin, but instead he

  (i) proved that L is in NP and

  (ii) picked a language L' already known to be NP-Complete and showed that $\mathrm{L}' \leqslant_{\mathrm{p}} \mathrm{L}$

Exercise: Reflect on why the strategy used by Karp works for proving that a language L is NP-Complete

The Cook-Levin Theorem

Karp's work popularized the notion of NP-Completeness and by using his strategy thousands of other problems have been proved to be to NP-Complete

Hence the fundamental importance of the work of Cook and Levin since they established the **first** NP-Complete problem (which could then be used in Karp's reductions)

Theorem 2.10 (Cook-Levin Theorem [Coo71, Lev73])

1. SAT is NP-Complete
2. $3SAT$ is NP-Complete

Before proving this theorem lets take a look at the problems SAT and $3SAT$

The SAT Problem (I)

A Boolean formula consists of variables $x_1, \dots x_n$and the logical operators AND ($\wedge$), NOT ($\neg$) and OR ($\vee$)

If φ is a Boolean formula over variables $x_1, \dots x_n$, and $z \in \{0,1\}^n$, then $\varphi(z)$ denotes the value of φ when the variables of φ are assigned the values in z (1 is true and 0 is false)

A formula φ is **satisfiable** if there there exists some assignment z such that $\varphi(z)$ is true, otherwise we say that it is **unsatisfiable**.

The formula $(x_1 \wedge x_2) \vee (x_2 \wedge x_3) \vee (x_3 \wedge x_1)$ with 3 variables, for instance, is satisfiable since the assignment $x_1 = 1$, $x_2 = 0$, and $x_3 = 1$ satisfies it

The SAT Problem (II)

- Any Boolean formula can be put in the **CNF** form (shorthand for Conjunctive Normal Form)

- A formula is in CNF if it is a conjunction of clauses, each clause is a disjunction of literals (variables and negation of variables). Example: $(x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee x_3 \vee \overline{x_4})$, where $\overline{x}$ denotes the negation of the variable x

- The SAT language consists of all satisfiable Boolean formulas in the CNF format

- The $3SAT$ language consists of all satisfiable formula in the **3CNF** form (CNF form in which all clauses have at most 3 literals)

Proving that SAT is NP-Complete

In order to show that SAT is NP-Complete we have to show that SAT is in NP and that SAT is NP-Hard.

The most difficult part of the proof that SAT is NP-Complete is to prove that SAT is NP-hard, i.e, to prove that all problems in NP can be reduced to SAT (this proof will be done later). The proof that SAT is in NP is easy:

Exercise: Prove that SAT is in NP

Exercise: Assuming the fact that SAT has already been proved to be NP-Complete explain what could be a stategy to prove that $3SAT$ is NP-Complete