Teoria da Computação
2018/2

Álvaro Moreira `afmoreira@inf.ufrgs.br`

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

Contents

Boolean Functions/Decision Problems/Languages

- A complexity class is a set of functions/problems/languages that can be computed within given resource bounds
- For convenience, usually only Boolean functions are considered (i.e functions from $\{0,1\}^*$ to $\{0,1\}$)
- Boolean functions define decision problems or languages
- We identify a Boolean function f from $\{0,1\}^*$ to $\{0,1\}$ with the language $L_f \subseteq \{0,1\}^*$

  $$L_f = \{x \mid f(x) = 1\}$$

  the elements of the language are the strings x in $\{0,1\}^*$ for which the Boolean function returns "true"(i.e for which $f(x) = 1$)

Example (I)

The *dinner party* problem consists in inviting to a party the largest number of your friends that get along to each other.

Representing the possible invitees to a dinner as the vertices of a graph G and placing an edge between any two people that don't get along, the dinner party computational problem becomes the problem of finding a **maximum sized independent set** of G

Finding the maximal independent set (the largest set of invitees that get along) is an NP-Hard optimization problem (more about NP-Hard latter)

Certainly that is the problem of interest in practice, but for Complexity it is convenient to deal with the Boolean function/decison problem/language version of the original problem

Example (II)

In its decision version the problem consists in deciding if, given a graph G and a positive integer k, if there is an independent set of size at least k

The Boolean function $f_{\mathrm{INDSET}} : \{0,1\}^* \to \{0,1\}$ associated to the problem is

$$f_{\mathrm{INDSET}}(G, \boxed{k}) = \begin{cases} 1 & \text{if } \exists S \subseteq V(G) \text{ s.t. } |S| \geq \boxed{k} \wedge \ \forall u, v \in S, \overline{uv} \notin E(G) \\ 0 & \text{otherwise} \end{cases}$$

The language $L_{\mathrm{INDSET}} \subseteq \{0,1\}^*$ associated to the Boolean function is

$$L_{\mathrm{INDSET}} = \{\langle G, k\rangle \mid \exists S \subseteq V(G) \text{ s.t. } |S| \geq k \ \wedge \ \forall u, v \in S, \overline{uv} \notin E(G)\}$$

The decision problem associated with the language is the following:

given a graph G and a positive number k, does $\langle G, k\rangle \in L_{\mathrm{INDSET}}$?

The Class NP (I)

- We all know that there is a big difference between solving a problem from scratch and verifying a given solution.
- The usual explanation: solving requires creativity, while verifying is much more easier since someone else has already done the creative work
- The complexity class P captures the set of problems that can be **efficiently solved**
- The complexity class NP captures the set of problems whose **solutions can be efficiently verified**.

Note: NP – Complete problems are the "hardest"problems in NP

The Class NP (II)

A **solution** to a problem is **efficiently verifiable** if it can be **verified** in polynomial time that it is indeed a solution.

Since a TM can only read one bit in a step, the length of a presented solution/certificate can be at most polynomial in the length of the input

Definition: (The class NP) *A language* $L \subseteq \{0,1\}^*$ *is in* NP *if there is a polynomial* $p : \mathbb{N} \to \mathbb{N}$ *and a polynomial time TM* V *(called verifier for* L*) s.t for every* $w \in \{0,1\}^*$

$$w \in L \quad \Leftrightarrow \quad \exists c \in \{0,1\}^{p(|w|)} \text{ s.t. } V(\langle w, c \rangle) \text{ stops in } \textit{accept} \text{ state}$$

If $w \in L$ *and* $c \in \{0,1\}^{p(|w|)}$ *are such* $V(\langle w, c \rangle)$ *stops with accept then we call* c *a* ***certificate*** *for* w *(w.r.t. language* L *and TM* V*)*

Note: Clearly $P \subseteq NP$. The simplest way to prove this fact is considering the definition of NP as the class of problems decided by a nondeterministic TM in polynomial time.

Exercise: Prove that $P \subseteq NP$.

Example - INDSET is in NP (I)

We have to show that there is a certificate whole length is polynomial in the length of the iout w, and that there is a polynomial time TM V such that for every string $w \in \{0,1\}^*$ encoding a pair (G, k)

$$w \in L_{\mathrm{INDSET}} \quad \Leftrightarrow \quad \exists c \in \{0,1\}^{p(|w|)} \text{ s.t. } V(w, c) = 1$$

A certificate is a string c encoding a list of vertices of G such that for any two vertices u, v in this list there are no edges connecting them.

Certainly there is a polynomial p such that $c \in \{0,1\}^{p(|w|)}$ because the length of c is smaller then the length of the input string w (which encodes (G, k))

If n is the number of vertices of G then it is necessary $\log n$ bits to encode each vertice, hence a list c of k vertices can be encoded using $O(k \log n)$. Thus c is a string of at most $O(n \log n)$ which is polynomial in the size of the input w

Example - INDSET is in NP (II)

We still have present a TM M that

- (i) checks if c encodes a list of size at least k of vertices not connected by any edge in G, and
- (ii) does that checking in polynomial time

It is not necessary to formally define every detail of such a TM, it is enough to give a high level description of its behaviour and argue that if does the checking in a time bounded by a polynomial in the length of its input

Other NP problems (I)

- Linear Programming: Given a list of m linear inequalities with rational coefficients over n variables $u_1, \ldots u_n$, **decide if there is** an assignment of rational numbers to the variables that satisfies all the inequalities.

- Composite numbers (or non-primality): Given a number N **decide** if N is a composite (i.e., non-prime) number.

- Connectivity: Given a graph G and two vertices s, t in G, **decide** if s is connected to t in G.

Exercise: Prove that connectivity, composite-numbers, and linear-programming are in NP.

Other NP problems (II)

- Traveling Salesperson: Given a set of n nodes, $\binom{n}{2}$ numbers $d_{i,j}$ denoting the distances between all pairs of nodes, and a number k, **decide** if there is a closed circuit that visits every node exactly once and has total distance of at most k.

- Subset sum: Given a list of n numbers $A_1, \ldots A_n$ and a number T, **decide if there is** a subset of the numbers that sums up to T.

- 0,1 Integer Programming: Given a list of m linear inequalities with rational coefficients over n variables $u_1, \ldots u_n$ **decide** if there is an assignment of zeroes and ones to $u_1, \ldots u_n$ satisfying the inequalities.

Exercise: Prove that the problems above are in NP.

Other NP Problems (III)

- Graph Isomorphism: Given two $n \times n$ adjacency matrices M_1, M_2, decide if M_1 and M_2 define the same graph, up to renaming of vertices. The certificate is the permutation $\pi : [n] \to [n]$ such that M_2 is equal to M_1 after reordering M_1's indices according to π
- Factoring: Given three numbers N, L, U **decide** if N has a prime factor p in the interval $[L, U]$. The certificate is the factor p.

Note: Graph isomprphism and Factoring are examples of NP problems which are not known to be in P nor $NP-Complete$.

Note: Traveling salesperson, Subset sum, and Integer programming are examples of NP problems which are not known to be in P. They are $NP-Complete$ problems

Relation between NP and P

- Recall that:

  Definition: (The class P) $\quad \mathsf{P} = \bigcup_{k \geq 1} \mathsf{DTIME}(n^k)$.

- Now let's define

$$\mathsf{EXP} = \bigcup_{k \geq 1} \mathsf{DTIME}(2^{n^k})$$

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

$$\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{EXP}$$

Relation between P and NP (II)

Proof of $P \subseteq NP$:

Exercise given. For this proof consider using the definition of NP in terms of nondeterministic TMs.

Proof of $NP \subseteq EXP$:

Suppose $L \in NP$ and suppose TM V and the polynomial p are as in the previous definition of NP. We can **decide** L, i.e. we can **decide** if a string $w \in L$, by enumerating **all** possible strings u of size at most $p(|w|)$ (i.e all $u \in \{0, 1\}^{p(|w|)}$), and by using V to check whether one of these strings is a **certificate** for the input w.

If there is such a string then $w \in L$ otherwise if all strings u are tested by V and none of them is a certificate, the conclusion is that $w \notin L$.

Relation between P and NP (III)

Note that the machine V used above runs in polynomial time but it might have to check all possible strings u.

The size of each u is at most polynomial in the size n of the input w. In other words the size of each u is $O(n^k)$ for some $k > 1$.

Since the number of strings u of size $O(n^k)$ in $\{0,1\}^*$ is $2^{O(n^k)}$ (exponential), and since V might have to check all of them, the number of times that we might have to run V is exponential.

Hence $L \in EXP$ $\square$

Reduction (I)

- The independent set problem is an example of a problem at least as hard as any other language in NP
- That means that, if the independent set problem has a polynomial-time algorithm then so do all the problems in NP.
- This property is called NP-hardness. We say that independent set is an $\mathsf{NP-Hard}$ problem.
- 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 time reduction** (which is a mapping reduction done in polynomial time - also known as Karp reduction)

Reduction (II)

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

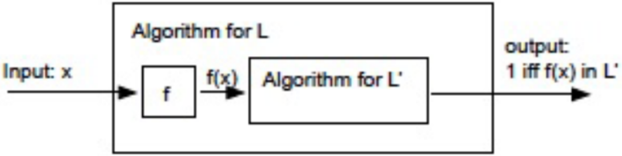
- *We say that a language* $L \subseteq \{0,1\}^*$ *is* ***polynomial-time Karp reducible*** *to a language* $L' \subseteq \{0,1\}^*$, *denoted by* $L \leq_p L'$, *if there is a polynomial-time computable function* $f : \{0,1\}^* \to \{0,1\}^*$ *such that for every* $w \in \{0,1\}^*$, $w \in L$ *iff*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

- *We say that* L' *is* ***NP-hard*** *if* $L \leq_p L'$ *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 $L \leq_p L'$ and if $L' \in P$ (i.e. the algorithm for L' in the figure above is polynomial) then $L \in P$

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:

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

- 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 w, $f_2 \circ f_1$ takes polynomial-time to compute $(f_2 \circ f_1)(w)$ (see observation before before the theorem above), and

  (ii) since $w \in L$ iff $f_1(w) \in L'$ and since $f_1(w) \in L'$ iff $f_2(f_1(w)) \in L''$, we have that $w \in L$ iff $f_2(f_1(w)) \in L''$ (i.e. $w \in L$ iff $(f_2 \circ f_1)(w)) \in L''$)

Exercise: Prove 2 and 3 of the theorem above.