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

The class NP

Attempts to avoid brute-force algorithms in the solution of certain important and useful problems have failed

polynomial time algorithms that solve them are not know to exist

What is the reason for such a failure? The answer for that question is not known

Maybe these are problems that **cannot** be solved in polynomial time. Or maybe they do have a polynomial solution waiting to be discovered

One remarkable discovery however showed that the complexity of many of these problems are connected: **a polynomial algorithm that solves one of them can be used to solve an entire class of problems**

The class NP

A Hamiltonian path in a directed graph G is a direct path that goes through each node exactly one.

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

We can easily define bute-force exponential time algorithm for HAMPATH: consider all possible paths adding a check to see if a potential path is Hamiltonian or not

No one knows if a HAMPATH can be solved in polynomial time

But this problem has a feature called **polynomial verifiability**:

If such a Hamiltonian path is discovered (perhaps using an exponential time solution) then to **verify that this path is indeed Hamiltonian can be done in polynomial time**

The class NP

A number is **composite** if it is the product of two integers greater than 1 (i.e. a number is composite if it is not a prime)

$\text{COMPOSITE} = \{x \mid x = p.q, \text{for integers } p,\ q > 1\}$

This problem can be decided in polynomial time and it is also polynomially verifiable. All we need is a divisor for that number.

Some problems may not be polynomially verifiable . Consider the $\overline{\text{HAMPATH}}$ problem, the complement of HAMPATH

We can decide in exponential time if a graph has no Hamiltonian path ...

But no one knows a way to **check** this in polynomial (what there is to check in polynomial time in this case?)

The class NP

Definition: *7.18 A* ***polynomial time verifier*** *for a language* A *is a polynomial time algorithm* V *such that*

$$A = \{w \mid V \text{ accepts } \langle w, c \rangle \text{ for some certificate } c\}$$

The size of the certificate c has to be polynomial in the size of w so the verifier can run in polynomial time.

Definition: *7.19* NP *is the class of languages that have polynomial time verifiers.*

The languages HAMPATH and COMPOSITE are both in the class NP.

HAMPATH is in NP

To prove that HAMPATH is in NP we have to think what would be a certificate for a string $\langle G, s, t \rangle \in$ HAMPATH...

The certificate c in this case is a (codification of) path from s to t in G that is Hamiltonian.

First observe that the length of this path/certificate c certainly is polynomial with respect to the length of $\langle G, s, t \rangle$

And we can easily check in polynomial time that it is indeed a Hamiltonian path in the graph G connecting s and t

In other words there is a polynomial time verifier V for HAMPATH, hence, HAMPATH is in NP

COMPOSITE is in NP

What would be a certificate for a number $x \in$ COMPOSITE ?

A certificate that a number x is composite can be just another number that is one of the divisors of x (different than 1 and different than x)

The length of this certificate c is celarly polynomial in the length of x

And to check that this number c is a divisor for the number x can be done in polynomial time by the verifier

Observe that COMPOSITE is a member of class P but it also can be verified in polynomial time so it is also a member of NP

All problems that can be decided in polynomial time can be verified in polynomial time:

$$P \subseteq NP$$

Another way to define NP

The name NP comes from **nondeterministic polynomial time**:

Theorem: A language is in NP iff it is decided by some **nondeterministic** polynomial time TM.

Proof: (i) For the $\Rightarrow$ direction: let A be a language in NP and define a nondeterministic TM N that decides A in polynomial time. Since $A \in NP$ then there is a polynomial time verifier V for it. Assume that V is a TM that runs in time n^k and define N as follows:

N = *"On input w of length n:*

1. *Nondeterministically select string c of length at most* n^k
2. *Run V on* $\langle w, c \rangle$
3. *If V accepts then accept; otherwise reject"*

Another way to define NP

...

(ii) For the $\Leftarrow$ direction: assume that A is decided by a nondeterministic TM N and construct a polynomial time verifier V (that has to be deterministic):

V = *"On input* $\langle w, c \rangle$*:*

1. *Simulate* N *on input* w *treating each symbol of* c *as the result of a nondeterministic choice made by* N
2. *If this branch of* N*'s computation accepts, then accept; otherwise reject"*

□

Definition: $\mathrm{TIME}(t(n)) = \{ L \mid L \text{ can be decided by an } O(t(n)) \text{ time DTM} \}$

$$P = \bigcup_k \mathrm{TIME}(n^k)$$

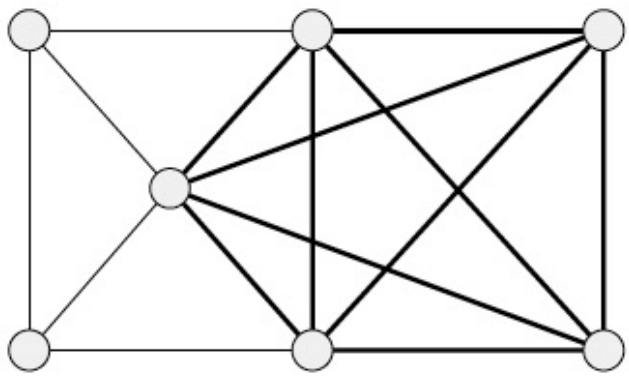
Definition: $\mathrm{NTIME}(t(n)) = \{ L \mid L \text{ can be decided by an } O(t(n)) \text{ time ND-TM} \}$

$$NP = \bigcup_k \mathrm{NTIME}(n^k)$$

More examples of problems in NP

A **clique** in an undirected graph is a subgraph where every two nodes are connected by an edge.

A **k-clique** is a clique that has k nodes. The graph below has a 5-clique:



The clique problem: given a graph G and a positive number k determine if G has a k-clique.

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

CLIQUE is in NP

Theorem: CLIQUE is in NP.

Proof: The certificate c is the clique. Certainly its size is polynomial in the size of the graph G. We have to show that we can construct a polynomial time verifier V for CLIQUE:

V = *"On input* $\langle\langle G, k\rangle, c\rangle$*:*

1. *Test if* c *is a subgraph of* G *with* k *nodes*
2. *Test if every two nodes of* c *are connected by edges of* G
3. *If both tests return true then accept; otherwise reject"*

□

Alternative proof: construct a ND TM N that decides CLIQUE in polynomial time:

N = *"On input* $\langle G, k\rangle$*:*

1. *Nondeterministically select a subset* c *of* k *nodes of* G
2. *Test if every two nodes of* c *are connected by edges of* G
3. *If yes then accept; otherwise reject"*

□

SUBSET is in NP

We give a multiset S of numbers $x_1, \ldots x_n$ and another number t. Determine if S contains a submultiset that adds up to t.

$$\text{SUBSET} = \{\langle S, t\rangle \mid S = \{x_1, \ldots x_n\}, \text{ and for some } \{y_1 \ldots y_m\} \subseteq \{x_1, \ldots x_n\}, \text{we have that } \Sigma y_i = t\}$$

Theorem: SUBSET is in NP.

Proof: The certificate is the subset. . We have to show that we can construct a polynomial time verifier V for SUBSET:

$V =$ *"On input* $\langle\langle S, t\rangle, c\rangle$*:*

1. *Test if* c *is a multiset of numbers that sum to* t
2. *Test if* S *contains all numbers in* c
3. *If both tests return true then accept; otherwise reject"* □

The P versus NP question

P is the set of languages for which membership that can be decided quickly

NP is the set of languages for which membership that can be verified quickly

HAMPATH, CLIQUE, and SUBSET are examples of languages that are in NP but are not known to be in P

We are also unable to prove the existence of a single language that is in NP but it is not in P

This is the famous unsolvable question $P = NP?$. If theses classes were equal, any polynomially verfed problem could also be polynomially decided

Many people have tried to find polynomial time algorithms for problems in NP that are not known to be in P and have failed

Researchers also have tried to prove that $P \neq NP$, in which case no fast algorithms would exist for problems in NP today solved only with brute-force in exponentail time

NP-completeness

One important advance in the P versus NP question came with a discovery made by Cook and Levin in the 70s

The showed that certain problems in NP are the *hardest* to solve and they are equally hard, in the sense that, if a polynomial time algorithm for one of them is found then all problems in NP would be solvable in polynomial time

These problems are called NP – Complete and they are important both for theory and for practice

On the theory side, researches working on the P versus NP question can focus on the NP – Complete problems either trying to solve one of them in polynomial time to proof that $P = NP$. Or trying to prove the impossibility of solving them in polynomial time and then proving that $P \neq NP$

On the practical side if the problem we are interested is NP – Complete we know that we cannot waist our time looking for a polynomial solution for it