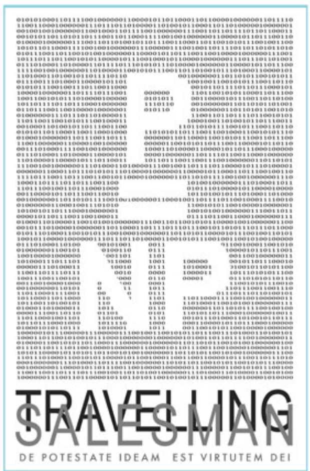
Limits of Computation

18 - The One Million Dollar Question
Bernhard Reus

A Complexity Class for Problems from Last Time

THIS TIME

- last time we have seen many problems which have not been shown (yet) to be in P but ...
- ... we can show that their solutions can be checked in polynomial time
- ... which leads to a new complexity class.
- *BIG QUESTION*: does this new class really contain more problems than P?



www.travellingsalesmanmovie.com

Recall from last time

- We have seen several problems that ...
- ... can be decided easily by *"brute-force"* search which unfortunately does not have polynomial time complexity so they do not prove that the problems are in P.
- But it can be checked in polynomial time whether a *potential solution is* actually a solution.
- we will now introduce a *new class* to capture those problems using this idea.

New complexity classes

Definition 18.1 (Verifier). A $\mathtt{L}$-*verifier* for a problem $A \subseteq \{0,1\}^*$ is a $\mathtt{L}$-program p (that always terminates) such that

certificate c for a d

$$A = \{\, d \in \{0,1\}^* \mid [\![p]\!]^{\mathtt{L}}(d,c) = \text{true for some } c \in \{0,1\}^* \,\}$$

IMPORTANT: time is considered only in the size of input d, not the certificate

Definition 18.2. 1. The class of *problems* $\mathtt{L}$*-verifiable in time* f is:
$\mathbf{NTIME}^{\mathtt{L}}(f) = \{A \subseteq \{0,1\}^* \mid A \text{ has an } \mathtt{L}\text{-verifier } p \in \mathtt{L}^{time(f)}\}$

2. $\mathbf{NP}^{\mathtt{L}}$ is the class of problems that have polynomial time $\mathtt{L}$-verifiers, or in other words, $\mathbf{NP}^{\mathtt{L}} = \{A \subseteq \{0,1\}^* \mid A \text{ has an } \mathtt{L}\text{-verifier } p \in \mathtt{L}^{ptime}\}$
3. $\mathbf{NLIN}^{\mathtt{L}} = \{A \subseteq \{0,1\}^* \mid A \text{ has an } \mathtt{L}\text{-verifier } p \in \mathtt{L}^{lintime}\}$
4. $\mathbf{NEXP}^{\mathtt{L}} = \{A \subseteq \{0,1\}^* \mid A \text{ has an } \mathtt{L}\text{-verifier } p \in \mathtt{L}^{exptime}\}$

Why the name **NP**?

- **NP** stands for **n**ondeterministically **p**olynomial
- the reason for this is the following:

Theorem 18.1. *$A \in$ **NP**^{TM} if, and only if, A is accepted by a nondeterministic Turing machine (*`NTM`*) with polynomial time bound. Similarly, $A \in$ **NLIN**^{TM} if, and only if, A is accepted by a nondeterministic Turing machine (*`NTM`*) with linear time bound and $A \in$ **NEXP**^{TM} if, and only if, A is accepted by a nondeterministic Turing machine (*`NTM`*) with exponential time bound*

explanation of "accepting" follows

Nondeterministic Programs

- for `TM` add a construct

$$\ell\texttt{:}\quad \texttt{goto}\ \ell'\ \texttt{or}\ \ell''.$$

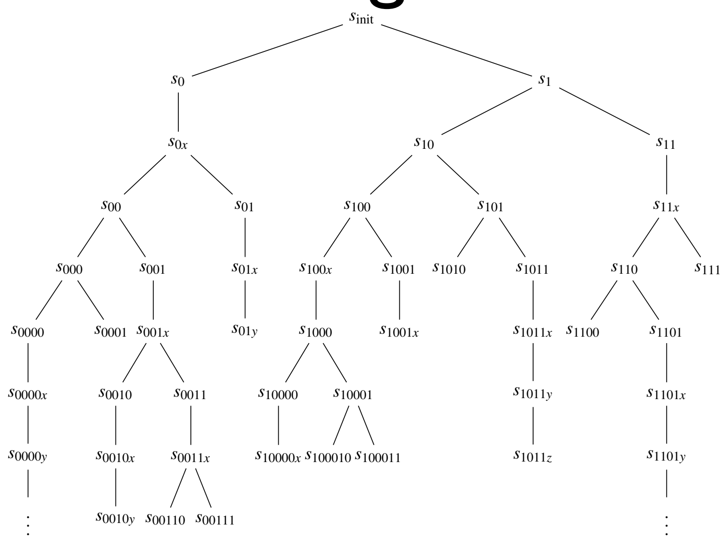
- the semantics of which is

$$(\ell,\sigma) \to (\ell',\sigma) \quad \text{and} \quad (\ell,\sigma) \to (\ell'',\sigma)$$

- so program execution does not produce a transition sequence of states but a tree of states.
- for `WHILE` you can do this too: add a command

`choose C1 or C2`

Nondet. Program Runs



Nondet. Program Semantics

- semantics is not longer a function but a relation, relating input values and possible outputs:

$$[\![\_]\!]^{\mathtt{L}} \subseteq \text{L-data} \times \text{L-data}_{\perp}$$

- similarly, time measure for nondet. programs is now a relation:

$$\mathit{time\_nd}_p^{\mathtt{L}} \subseteq \text{L-data} \times \mathbb{N}_{\perp}$$

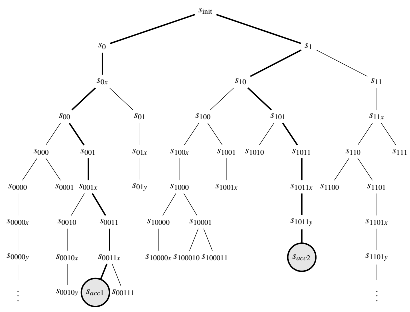
$\mathit{time}_p^{\mathtt{L}}(d) = \min\{\, t \mid (d,t) \in \mathit{time\_nd}_p^{\mathtt{L}} \text{ such that } p \text{ accepts input } d \,\}$

can't use "decided by procedure" due to non-determinism

Nondeterministic Programs

Definition 18.3 (Accepted set of a nondeterministic program). A nondeterministic L-program p *accepts* input d if $(d, \text{true}) \in [\![p]\!]^{\mathtt{L}}$. The set $Acc(p) \subseteq$ L-data, called the set of data *accepted by* p, is defined as follows:

$$Acc(p) = \{\, d \in \text{L-data} \mid p \text{ accepts } d \,\}$$



Example: accepting paths

Robustness of NonDet. Computation

- we can show that with nondeterministic programs we cannot solve more problems...
- ... as they can be simulated by deterministic ones using exponential slow-down.
  Proof: *write an interpreter that does dove-tailing for each alternative at a choice command.*
- Evidence for Church-Turing Thesis

Results about **NP**

Results NP

$\boldsymbol{P} \subseteq \boldsymbol{NP}$.

Why?

Robustness

Theorem 18.2. *Aside from data-encoding issues*

$$\boldsymbol{NP}^{TM} = \boldsymbol{NP}^{SRAM} = \boldsymbol{NP}^{GOTO} = \boldsymbol{NP}^{WHILE}$$

Problems are in NP

Theorem: TSP, Graph-Colouring, 0-1Knapsack, and Integer Linear Programming are in NP.

- Write a *verifier* that takes as input the instance (graph etc), and quality measure *K*, and as *certificate* a candidate solution *(a tour, colouring, packing, variable assignment etc)*.
- It must verify that the candidate solution is actually a solution for the given instance (scenario) of quality *K* or better.
- Check that the verifier runs in polynomial time in the size of the input scenario.

Check e.g. for TSP

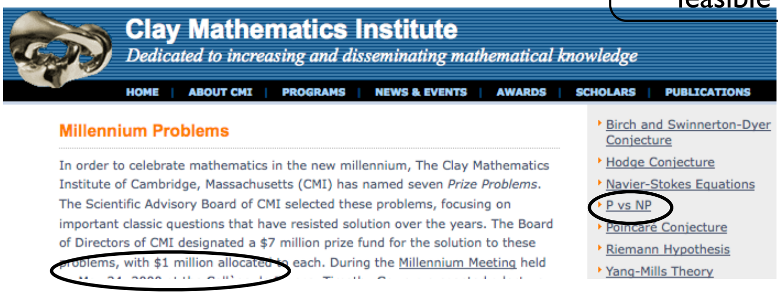
- The certificate here is a list of cities (path/tour).
- The verifier needs to add up the distance of the edges between the cities of the path in given order and compare them with *K*.

must also check that all nodes are visited

- This can be done in time $\mathcal{O}(|V|)$, so polynomially in the size of the graph.

P = NP ?

are NP problems “feasible” after all?



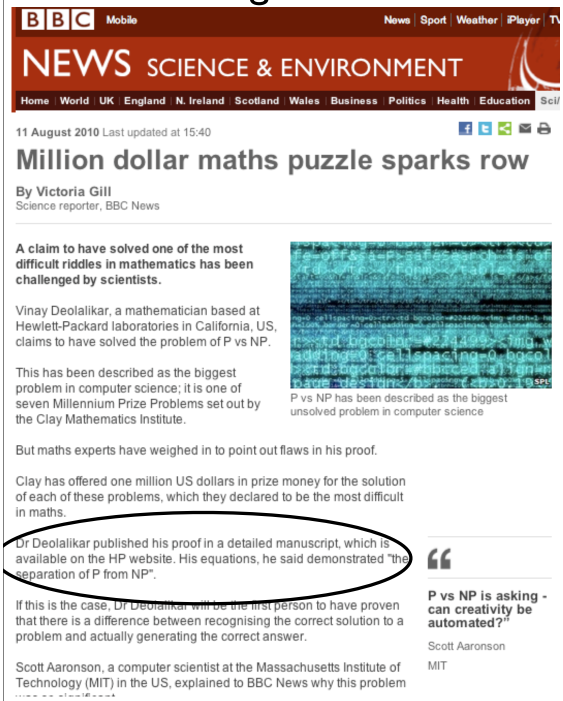
The most famous BIG OPEN PROBLEM in Theoretical Computer Science

P = NP ?

- Evidence so far suggests that this is not the case.
- Here is what Scott Aaronson (MIT) says:

> *“If **NP** problems were feasible, then mathematical creativity could be automated. The ability to check a proof would entail the ability to find one. Every Apple II, every Commodore, would have the reasoning power of Archimedes or Gauss.”*

11 August 2010



BBC Mobile | News | Sport | Weather | iPlayer | TV

NEWS SCIENCE & ENVIRONMENT

Home | World | UK | England | N. Ireland | Scotland | Wales | Business | Politics | Health | Education | Sci/

11 August 2010 Last updated at 15:40

Million dollar maths puzzle sparks row

By Victoria Gill
Science reporter, BBC News

A claim to have solved one of the most difficult riddles in mathematics has been challenged by scientists.

Vinay Deolalikar, a mathematician based at Hewlett-Packard laboratories in California, US, claims to have solved the problem of P vs NP.

This has been described as the biggest problem in computer science; it is one of seven Millennium Prize Problems set out by the Clay Mathematics Institute.

P vs NP has been described as the biggest unsolved problem in computer science

But maths experts have weighed in to point out flaws in his proof.

Clay has offered one million US dollars in prize money for the solution of each of these problems, which they declared to be the most difficult in maths.

Dr Deolalikar published his proof in a detailed manuscript, which is available on the HP website. His equations, he said demonstrated "the separation of P from NP".

If this is the case, Dr Deolalikar will be the first person to have proven that there is a difference between recognising the correct solution to a problem and actually generating the correct answer.

Scott Aaronson, a computer scientist at the Massachusetts Institute of Technology (MIT) in the US, explained to BBC News why this problem

“P vs NP is asking - can creativity be automated?”

Scott Aaronson
MIT

Recent proof attempt of **P=NP** failed!

END

Next time:
which problems in NP are actually really “hard”?