Great Theoretical Ideas In Computer Science

Anupam Gupta
Lecture 28  Dec 7th, 2006

CS 15-251  Fall 2006
Carnegie Mellon University

Complexity Theory: the P vs NP question
The Clay Mathematics Institute
Millennium Prize Problems
1. Birch and Swinnerton-Dyer Conjecture
2. Hodge Conjecture
3. Navier-Stokes Equations
4. P vs NP
5. Poincaré Conjecture
6. Riemann Hypothesis
7. Yang-Mills Theory

http://www.claymath.org/millennium/
The P versus NP problem

Is perhaps one of the biggest open problems in computer science (and mathematics!) today.

(Even featured in the TV show NUMB3RS)

But what is the P-NP problem?
SUDDOKU

\[
\begin{array}{ccc}
2 & 3 & 8 \\
3 & 4 & 9 \\
6 & 7 & 9 \\
\hline
3 & 5 & 6 \\
4 & 9 & 5 \\
9 & 6 & 2 \\
\hline
8 & 7 & 3 \\
9 & 8 & 4 \\
3 & 1 & 9 \\
\end{array}
\]
<table>
<thead>
<tr>
<th>2</th>
<th>4</th>
<th>7</th>
<th>8</th>
<th>1</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>9</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>2</td>
<td>1</td>
<td>9</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6</th>
<th>7</th>
<th>1</th>
<th>9</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>8</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>1</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>9</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8</th>
<th>2</th>
<th>5</th>
<th>9</th>
<th>6</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>9</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8</th>
<th>2</th>
<th>5</th>
<th>9</th>
<th>6</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>9</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8</th>
<th>2</th>
<th>5</th>
<th>9</th>
<th>6</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>9</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

SUDOKU

3x3x3
Suppose it takes you \( S(n) \) time to solve \( n \times n \times n \) Sudoku.

\[ V(n) \] to verify the solution

**Fact:** \[ V(n) = O(n^2 \times n^2) \]

**Q:** is there some constant such that \( S(n) \leq [V(n)]^{\text{constant}} \)?
**SUDOKU**

P-vs-NP problem

Does there exist an algorithm for $n \times n \times n$ Sudoku that runs in time $p(n)$ for some polynomial $p(.)$?
The P versus NP problem (informally)

Is proving a theorem much more difficult than checking the proof of a theorem?

Loosely, that is what the P-vs-NP question asks.
Let's start at the beginning...
Given a graph $G=(V,E)$, a path that visits all the nodes without visiting any node twice.
**THE PROBLEM “HAM”**

**Input:** graph $G = (V,E)$

**Output:**
- YES, if $G$ has a Hamilton cycle
- NO, if $G$ has no Hamilton cycle.

**THE SET “HAM”**

$HAM = \{ \text{graphs } G \mid G \text{ has a hamilton cycle} \}$
**Circuit SAT**

**Input**: a combinatorial circuit $C$ with one output

**Output**: YES, if $C$ is satisfiable

NO, if $C$ is not satisfiable
Circuit Sat

\((x_1 \land x_2) \land (\neg x_2)\)

\((x_1 \land x_2) \land (\neg x_2 \lor x_3)\)
THE SET "SAT"

SAT = \{ \text{all satisfiable circuits } C \}

any \ C \in \text{SAT} \quad \text{has at least one satisfying assignment of } T/F \text{ to its variables.}
**Bipartite Matching**

**Input:** a bipartite graph $G = (U, V, E)$

**Output:**
- **YES** if $G$ has a perfect matching
- **NO** if $G$ has no perfect matching
The set "Bi-Match"

\( \text{Bi-Match} = \{ \text{all bipartite graphs } G \text{ that have a perfect matching} \} \)
SUDOKU

Input: \( n \times n \times n \) Sudoku instance

Output: YES: this Sudoku has a solution
        NO: has no solution

The set "SUDOKU"

\[ \text{SUDOKU} = \{ \text{all solvable Sudoku instances} \} \]
Decision vs Search Problems

Decision Problem
YES / NO
Does $G$ have a Hamilton cycle?

Search Problem
Find a Hamilton cycle in $G$ if one exists, else return NO.
Reducing Search to Decision

Given an algorithm for decision Sudoku, devise an algorithm to find a solution.

Idea:
Fill in one-by-one and use decision algorithm.

```
2  3  8  5
3  4  5  9  8
8  9  7  3  4
6  7  9
9  8  1  7
5  6  9
3  1  9  2
4  6  5  8
2  9  3  1
```
REDUCING SEARCH TO DECISION

Given an algorithm for decision HAM, devise an algorithm to find the cycle

Idea:
One by one, find the edges in the cycle.
Reducing search to decision

Diagram of a network of nodes and connections.
Reducing Search to Decision

\[ x_1, x_2, x_3 \]
Reducing Search to Decision
SEARCH / DECISION PROBLEMS

We'll study decision problems
(almost the same as their search counterparts)

HAM
SAT
BG MATCH.

View both as problems and sets
Polynomial Time

and

The Class 'P' of Decision Problems
What is an efficient algorithm?

Is an $O(n)$ time algorithm efficient?

How about $O(n \log n)$?

$O(n^2)$?

$O(n^{10})$?

$O(n^{\log n})$?

$O(2^n)$?

$O(n!)$?

$O(2^{2^n})$?

polynomial time

$O(n^c)$ for some constant $c$

non-polynomial time
Does an algorithm running in $O(n^{100})$ time count as efficient?
We consider non-polynomial time algorithms to be inefficient.

And hence a necessary condition for an algorithm to be efficient is that it should run in poly-time.
Asking for a poly-time algorithm for a problem sets a (very) low bar when asking for efficient algorithms.

The question is: can we achieve even this?
Once we know that our favorite problems have polynomial time algorithms, we can then worry about making them run in $O(n \log n)$ or $O(n^2)$ time!

What about our favorite problems: SAT, HAM, BI MATCH?
The class $P$ defined in the 50’s.

The Intrinsic Computational Difficulty of Functions, Alan Cobham, 1964.


not the correct Alan Cobham

this is indeed Jack Edmonds

An explanation is due on the use of the words "efficient algorithm"... I am not prepared to set up the machinery necessary to give it formal meaning, nor is the present context appropriate for doing this... For practical purposes the difference between algebraic and exponential order is more crucial than the difference between [computable and not computable]...

It would be unfortunate for any rigid criterion to inhibit the practical development of algorithms which are either not known or known not to conform nicely to the criterion... However, if only to motivate the search for good, practical algorithms, it is important to realize that it is mathematically sensible even to question their existence.

Edmonds called them “good algorithms”
The Intrinsic Computational Difficulty of Functions, Alan Cobham, 1964.

For several reasons the class $P$ seems a natural one to consider. For one thing, if we formalize the definition relative to various general classes of computing machines we seem always to end up with the same well-defined class of functions. Thus we can give a mathematical characterization of $P$ having some confidence it characterizes correctly our informally defined class.

This class then turns out to have several natural closure properties, being closed in particular under explicit transformation, composition and limited recursion on notation (digit-by-digit recursion).

if $p(\ )$ and $q(\ )$ are polynomials, then $p(q(\ ))$ is also a polynomial
The class $P$

We say a set $L \subseteq \Sigma^*$ is in $P$ if there is a program $A$ and a polynomial $p()$ such that for any $x$ in $\Sigma^*$,

$A$ (input $x$) runs for at most $p(|x|)$ time and answers question “is $x$ in $L$?” correctly.
The class $P$

The class of all sets $L$ that can be recognized in polynomial time.

The class of all decision problems that can be decided in polynomial time.
Why are we looking only at languages $\subseteq \Sigma^*$?

What if we want to work with graphs or boolean formulas?
Requiring that \( L \subseteq \Sigma^* \) is not really restrictive, since we can encode graphs and Boolean formulas as strings of 0's and 1's. In fact, we do this all the time: inputs for all our programs are just sequences of 0's and 1's encoded in some suitable format.
Example 1:

$\text{CONN} = \{ \text{graph } G: G \text{ is a connected graph} \}$

Algorithm $A_1$:

If $G$ has $n$ nodes, then run depth first search from any node, and count number of distinct nodes you see. If you see $n$ nodes, $G \in \text{CONN}$, else not.

Time: $p_1(|x|) = \Theta(|x|)$. 
Languages/functions in P?

\{ HAM, Sudoku, SAT \} \quad \text{not known.}

BL-MATCH \quad O(|V| \cdot \sqrt{|E|})
Languages/functions in P?

\[ \text{Co-HAM} = \{ \text{all graphs } G \text{ that have no Hamilton cycle} \} \]

\[ G \in \text{Co-HAM} \iff G \not\in \text{HAM}. \]

\[ \text{HAM} \in \text{P} \iff \text{Co-HAM} \in \text{P} \]
Onto the new class, NP
Verifying Membership

Is there a short "proof" I can give you for:

\[ G \in \text{HAM} \ ? \]

\[ G \in \text{Bi-Match} \ ? \]

\[ C \in \text{SAT} \ ? \]

\[ G \in \text{Co-HAM} \ ? \]
A set \( L \in \text{NP} \)

if \( \exists \) an algorithm \( A \)

and a polynomial \( p(\cdot) \)

\( \forall x \in L \)

\( \exists y \) with \( |y| \leq p(|x|) \)

s.t. \( A(x,y) = \text{YES} \)

in \( p(|x|) \) time

\( \forall x' \notin L \)

\( \forall y' \) with \( |y'| \leq p(|x'|) \)

\( A(x',y') = \text{NO} \)

in \( p(|x|) \) time
Recall the class $P$

We say a set $L \subseteq \Sigma^*$ is in $P$ if there is a program $A$ and a polynomial $p()$

such that for any $x$ in $\Sigma^*$,

$A$ (input $x$) runs for at most $p(|x|)$ time and answers question “is $x$ in $L$?“ correctly.

can think of $A$ as “proving” that $x$ in $L$. 
The new class NP

We say a set \( L \subseteq \Sigma^* \) is in \( \text{NP} \) if there is a program \( A \) and a polynomial \( p() \) such that for any \( x \) in \( \Sigma^* \),

If \( x \in L \), there exists a "proof" \( y \) with \( |y| \leq p(|x|) \)
\( A(x, y) \) runs for \( \leq p(|x|) \) time
and answers "\( x \) is in \( L \)" correctly.

If \( x \notin L \), for all "proofs" \( y \)
\( A(x, y) \) answers "\( x \) not in \( L \)" correctly.

Verifier rejects all "fake" proofs
The class NP

Non-deterministic polynomial time.

The class of sets \( L \) for which there exist “short” proofs of membership (of polynomial length) that can “quickly” verified (in polynomial time).

Recall: A doesn’t have to find these proofs \( y \); it just needs to be able to verify that \( y \) is a “correct” proof.
Which languages are in NP?
For any $L$ in $P$, we can just take $y$ to be the empty string and satisfy the requirements.

Hence, every language in $P$ is also in $NP$. 
Languages/functions in NP?

HAM? ✔

SAT? ✔

BI-MATCH? ✔

SUDOKU? ✔

NON-hamiltonian ✔

CO-HAM? ?
Summary: P versus NP

Set L is in P if
membership in L can be decided in poly-time.

Set L is in NP if
each x in L has a short “proof of membership”
that can be verified in poly-time.

Fact: $P \subseteq NP$

Question: Does $NP \subseteq P$?
The P versus NP problem

If membership in $L$ can be verified in poly-time,
then can membership in $L$ be decided in poly-time?

$\implies \begin{cases} L \in \text{NP} \\ L \in \text{P} \end{cases}$
WHY CARE?
NP contains lots of problems we don’t know to be in P

Classroom Scheduling
Packing objects into bins
Scheduling jobs on machines
Finding cheap tours visiting a subset of cities
Allocating variables to registers
Finding good packet routings in networks
Decryption
...

Hence proving $P = NP$ would break cryptosystems
OK, OK, I CARE.

WHERE DO I BEGIN?
How can we prove that $NP \subseteq P$?

I would have to show that every set in NP has a polynomial time algorithm...

How do I do that?
It may take forever!
Also, what if I forgot one of the sets in NP?
Relax, Bonzo!

We can describe one problem $L$ in NP, such that if this problem $L$ is in P, then $\text{NP} \subseteq \text{P}$.

It is a problem that can capture all other problems in NP.
"THE HARDEST" SET IN NP.
SUDOKU

\[ \bigcup n \in \mathbb{Z} \{ \text{all solvable } n \times n \times n \text{ Sudoku instances} \} \]

SUDOKU has polynomial-time algorithm

\(\iff\)

All if NP does
The “hardest” sets in NP

Sudoku

3 Color

SAT

Clique

Independent-Set

HAM
How do you prove these are the hardest?
Theorem [Cook/Levin]:
SAT is one language in NP, such that if we can show SAT is in P, then we have shown \( \text{NP} \subseteq \text{P} \).

SAT is a language in NP that can capture all other languages in NP.

We say SAT is \text{NP-complete}.

Proof not difficult, but we defer the proof to 15-451 (Algos) or 15-453 (FLAC).
3-colorability

Circuit Satisfiability
SAT and 3COLOR: Two problems that seem quite different, but are substantially the same.

Also substantially the same as CLIQUE and INDEPENDENT SET.

If you get a polynomial-time algorithm for one, you can get a polynomial-time algorithm for ALL.
Any language in NP can be reduced (in polytime) to an instance of
\[ \text{Cook} | \text{Levin} \]
hence SAT is NP-complete
Any language in NP can be reduced (in polytime) to an instance of SAT, and SAT can be reduced (in polytime) to an instance of 3COLOR.
Hence:

Any language in NP can be reduced (in polytime to) an instance of SAT, which can be reduced (in polytime to) an instance of 3COLOR. Hence, 3COLOR is NP-complete.
To show a language $L$ is NP-complete

To show $L$ is NP-complete, we must

a) Show $L$ is in NP
b) Reduce all problems in NP to $L$

or

b) Reduce some NP-complete language (e.g. SAT) to $L$
(by exhibiting a function $F$ such that
$C$ is satisfiable $\Leftrightarrow F(C)$ is in $L$)
NP-complete problems

3COLOR, SAT, bin-packing, classroom scheduling, many other problems NP-complete. proofs follow above pattern.

NP-complete problems: “Hardest” problems in NP (if any of these problems in P, then all NP in P.)

NP-complete problems come up all the time
Many problems of practical importance are NP-complete
If $P \neq NP$, we will not have poly-time algorithms for them.
Right now, we don’t know.
Reference

Computers and Intractability: A guide to the Theory of NP-completeness

by Mike Garey and David Johnson
References

The NP-completeness Column: 1981-2005
by David Johnson
http://www.research.att.com/~dsj/columns/

The P versus NP problem
Official Description, Clay Institute
by Steve Cook
http://www.claymath.org/millennium/P_vs_NP/