The $P = NP$ Problem

Bristol Teachers’ Circle, June 26 2013

P.D. Welch, University of Bristol
The Königsberg Bridge Problem - Euler
A “graph” version

- As Euler observed: there can be no path crossing each bridge once, due to the number of “odd nodes.”
- A harder variation is due to Hamilton.
Hamiltonian Circuits
Q. Can each node be visited exactly once here?
Travelling Salesman Problem

Q Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?
Even to visit the lower 48 state capitals would require a brute force search of 48! possibilities: a 62 digit number. Computation time:
Even to visit the lower 48 state capitals would require a brute force search of 48! possibilities: a 62 digit number. Computation time:

10 trillion trillion times the current age of the universe.
Q: Kidney Donations:
A₁ needs a kidney and has a donor D₁ prepared to donate a kidney. But unfortunately D₁’s kidney is incompatible with A₁. Similarly A₂ is in the same situation: she has a donor D₂ but alas with an incompatible kidney. Of course, if by good luck D₁ is compatible with A₂ and D₂ with A₁, as long as they can meet up, they could swap.

- In 2011 60 surgeries performed 30 kidney transplants using a database set up for this matching.

Q: How can classify problems in general into “easy” and “hard”?
Alan Turing’s machine

- Essentially this simple conceptual device can simulate all discrete computational computing devices.
• We may thus use it to calibrate “problems”, by asking for an expression giving the total number of steps (or ”time”) taken for a solution to be output.
• We thus ask for a function that approximates the length of such computations in terms of its input. $n$ in $\mathbb{N}$. 
String Checking

- We give the machine the task of checking whether a string of 0, 1’s is of the form $0^k 1^k$. (Let $n = 2.k$.)

We could program this as follows:
1) Scan across the tape and *reject* (= output 0) if a 0 is found to the right of a 1; if not go back to the start position.
2) Repeat the following if both 0’s and 1’s remain on the tape: Scan across the tape deleting a single 0 and a single 1 each time.
3) if 0’s remain after all 1’s have been crossed off (or vice versa) then *reject*. Otherwise all 0’s and 1’s have been crossed off, so *accept* (= output 1).

The first stage takes roughly $n + n$ steps. During 2) it takes $n$ steps to scan the tape, but crosses off two digits each time, so it has at most $\frac{n}{2}$ scans to do here; so altogether about $\frac{n^2}{2}$ steps, and finally 3) another factor $n$.

- $n^2$ is the dominating term here, so we say this takes “$O(n^2)$” steps.
The definition of $P$ “polynomial time” problems

- Was this best possible? Well, no, but it leads to:

**Definition ($P$ - polynomial time problem)**

A problem is said to be in the class $P$ if (i) it is solvable by a Turing Machine $T$, and (ii) there is a polynomial expression $p(x) = a_kx^k + a_{k-1}x^{k-1} + \cdots a_1x + a_0$ so that on input of length $n$ $T$ solves the problem in $p(n)$ steps.

Note (1): often it is only the dominant term $n^k$ here that is most relevant.
Note (2): if a number is input as a binary string (as the machine computes in binary), a number $n$ has a binary expansion of length $\log_2 n$, so the input of $n$ has this length on the starting tape. Hence a number $n$ takes $\log_2 n$ steps to be ‘read’ at the start.

Problems known to be in $P$:
1 whether $p, q \in \mathbb{N}$ are relatively prime (the Euclidean algorithm);
2 given a graph and two nodes, whether there is a path between them.
What is $NP$?

• A solution to a problem may be easily verified as a correct solution even if it cannot be easily found.

Definition (Verifiers)

An algorithm for a problem is said to be a verifier for the problem, if it can check purported solutions to the problem. The input to such an algorithm consists of the string representing the original problem, $s$ say, together with a certificate string, $c$.

Definition (NP problem)

(i) If the algorithm can be implemented on a TM with input $s \bowtie c$ and it runs in polynomial time in the length of the input $s$ alone (with output 0/1 for "reject/accept"), it is called a polynomial time verifier.

(ii) A problem that has a polynomial time verifier algorithm is said to be in “$NP$”.

• Examples: primality, hamiltonian circuits, ..., in fact most of the previously mentioned problems.
Notice that if I have a poly. running time algorithm for the solution of a problem, that algorithm also counts as a verifier for the problem (we can always take the certificate $c$ as the empty string with no length). Thus

$$P \subseteq NP.$$ 

The question is:

$$?P \supseteq NP?$$

It is unknown whether every problem that can be verified in polynomial time, has a polynomial time running algorithm for its solution. Thus the Hamiltonian circuit problem can be verified in polynomial time, but no one has any idea whether there might be a polynomial time running algorithm for it.
A Case Study: Primality

• Primality until about 10 years ago was thought to be a hard problem: to test whether a number $N$ is prime did not seem to be a question that could be solved in time $N^k$ for a fixed $k$ and general $N$. A brute force method looks like you have to go through all numbers less than $\sqrt{N}$ and divide them into $N$ to test. This takes \textit{exponentially long} time, and so grows faster than any polynomial. Only in 2002 did three mathematicians find a polynomial time algorithm for checking primality. The algorithm does not produce any prime factors however, it merely answers the “Yes/No” question as to whether $N$ is prime or not.

• However most computer scientists and mathematicians believe that $P \neq NP$ but as yet we are unable to prove this.
A Case Study: Primality

• Primality until about 10 years ago was thought to be a hard problem: to test whether a number $N$ is prime did not seem to be a question that could be solved in time $N^k$ for a fixed $k$ and general $N$. A brute force method looks like you have to go through all numbers less than $\sqrt{N}$ and divide them into $N$ to test. This takes exponentially long time, and so grows faster than any polynomial. Only in 2002 did three mathematicians find a polynomial time algorithm for checking primality. The algorithm does not produce any prime factors however, it merely answers the “Yes/No” question as to whether $N$ is prime or not.

• However most computer scientists and mathematicians believe that $P \neq NP$ but as yet we are unable to prove this.
Let us say a problem $A$ (expressed using an alphabet of strings $\Sigma$) is *simpler than* problem $B$ (expressed using an alphabet of strings $T$), $A \leq_p B$ if there is a function, computable in polynomial time, $f : \Sigma \rightarrow T$ so that

$$ w \text{ is a solution to } A \iff f(w) \text{ is a solution to } B. $$

- Idea: If I can solve $B$ then I could solve $A$ as well because I have a translation procedure ($f$) for “pulling back” a solution from $B$ to one for $A$. Moreover this only requires polynomially many steps.

• Hardest Problem Idea: Are there “hardest problems” in NP? Such a problem is called *NP-complete*. If we could show such an NP-complete problem was actually in P then we should be able to show all NP-problems are in P, and thus $NP \subseteq P$ and the $P=NP$ problem is solved!

There are many NP-complete problems it turns out, but needless to say no one has been able to show that any are in P.
\textbf{NP-complete problems}

Let us say a problem \(A\) (expressed using an alphabet of strings \(\Sigma\)) is \textit{simpler than} problem \(B\) (expressed using an alphabet of strings \(T\)), \(A \leq_p B\) if there is a function, computable in polynomial time, \(f : \Sigma \rightarrow T\) so that

\[w\text{ is a solution to } A \iff f(w)\text{ is a solution to } B.\]

- Idea: If I can solve \(B\) then I could solve \(A\) as well because I have a translation procedure \((f)\) for “pulling back” a solution from \(B\) to one for \(A\). Moreover this only requires polynomially many steps.

- \textit{Hardest Problem Idea:} Are there “hardest problems” in NP? Such a problem is called \textit{NP-complete}. If we could show such an \textit{NP-complete} problem was actually in \(P\) then we should be able to show all \(NP\)-problems are in \(P\), and thus \(NP \subseteq P\) and the \(?P = NP?\) problem is solved!
**NP-complete problems**

Let us say a problem $A$ (expressed using an alphabet of strings $\Sigma$) is *simpler than* problem $B$ (expressed using an alphabet of strings $T$), $A \leq_p B$ if there is a function, computable in polynomial time, $f : \Sigma \rightarrow T$ so that

$$w \text{ is a solution to } A \Leftrightarrow f(w) \text{ is a solution to } B.$$  

- **Idea:** If I can solve $B$ then I could solve $A$ as well because I have a translation procedure ($f$) for “pulling back” a solution from $B$ to one for $A$. Moreover this only requires polynomially many steps.

- **Hardest Problem Idea:** Are there “hardest problems” in $NP$? Such a problem is called *NP-complete*. If we could show such an $NP$-complete problem was actually in $P$ then we should be able to show all $NP$-problems are in $P$, and thus $NP \subseteq P$ and the $P \neq NP$ problem is solved!

There are many $NP$-complete problems it turns out, but needless to say no one has been able to show that any are in $P$. 
Let us say a problem $A$ (expressed using an alphabet of strings $\Sigma$) is *simpler than* problem $B$ (expressed using an alphabet of strings $T$), $A \leq_p B$ if there is a function, computable in polynomial time, $f : \Sigma \rightarrow T$ so that $w$ is a solution to $A \iff f(w)$ is a solution to $B$.

- **Idea:** If I can solve $B$ then I could solve $A$ as well because I have a translation procedure ($f$) for “pulling back” a solution from $B$ to one for $A$. Moreover this only requires polynomially many steps.

- **Hardest Problem Idea:** Are there “hardest problems” in $NP$? Such a problem is called *NP-complete*. If we could show such an $NP$-complete problem was actually in $P$ then we should be able to show all $NP$-problems are in $P$, and thus $NP \subseteq P$ and the $P = NP$ problem is solved!

There are many $NP$-complete problems it turns out, but needless to say no one has been able to show that any are in $P$.
Notice that if you have an $NP$-complete problem, let us call it $Q$ say, then if we could prove that $Q \in P$ then we should have that $NP \subseteq P$ and the $P = NP$ problem would be solved.
Notice that if you have an $NP$-complete problem, let us call it $Q$ say, then if we could prove that $Q \in P$ then we should have that $NP \subseteq P$ and the $P = NP$ problem would be solved.

Theorem (Cook, Levin, c.1971)

There is an $NP$-complete problem
Notice that if you have an $NP$-complete problem, let us call it $Q$ say, then if we could prove that $Q \in P$ then we should have that $NP \subseteq P$ and the $P = NP$ problem would be solved.

**Theorem (Cook, Levin, c.1971)**

*There is an NP-complete problem*

In fact it turns out that there are many $NP$-complete problems:

- The Travelling Salesperson; Hamiltonian Circuits
NP-complete problems, contd.

- Notice that if you have an $NP$-complete problem, let us call it $Q$ say, then if we could prove that $Q \in P$ then we should have that $NP \subseteq P$ and the $P = NP$ problem would be solved.

**Theorem (Cook, Levin, c.1971)**

*There is an NP-complete problem*

In fact it turns out that there are many $NP$-complete problems:

- The Travelling Salesperson; Hamiltonian Circuits

- The Subset-Sum Problem: *Given a set of numbers $A = \{n_1, \ldots, n_k\}$ and a target number $M$ is there a subset $\{m_1, \ldots m_l\} \subseteq A$ with $m_1 + \cdots + m_l = M$?*
NP-complete problems, contd.

- Notice that if you have an NP-complete problem, let us call it $Q$ say, then if we could prove that $Q \in P$ then we should have that $NP \subseteq P$ and the $P = NP$ problem would be solved.

**Theorem (Cook, Levin, c.1971)**

*There is an NP-complete problem*

In fact it turns out that there are many $NP$-complete problems:

- The Travelling Salesperson; Hamiltonian Circuits

- The Subset-Sum Problem: *Given a set of numbers $A = \{n_1, \ldots, n_k\}$ and a target number $M$ is there a subset $\{m_1, \ldots, m_l\} \subseteq A$ with $m_1 + \cdots + m_l = M$?*

- The Kidney Matching Problem, Sudoku . . .
Notice that if you have an \( \mathsf{NP} \)-complete problem, let us call it \( Q \) say, then if we could prove that \( Q \in \mathsf{P} \) then we should have that \( \mathsf{NP} \subseteq \mathsf{P} \) and the \( \mathsf{P} = \mathsf{NP} \) problem would be solved.

**Theorem (Cook, Levin, c.1971)**

*There is an \( \mathsf{NP} \)-complete problem*

In fact it turns out that there are many \( \mathsf{NP} \)-complete problems:

- The Travelling Salesperson; Hamiltonian Circuits
- The Subset-Sum Problem: Given a set of numbers \( A = \{n_1, \ldots, n_k\} \) and a target number \( M \) is there a subset \( \{m_1, \ldots, m_l\} \subseteq A \) with \( m_1 + \cdots + m_l = M \)?
- The Kidney Matching Problem, Sudoku . . .
- Determining a computer user’s secret key from her public key.
Richard Kaye proved in 2000 that the Windows game was an \( NP \)-complete problem.
Richard Kaye proved in 2000 that the Windows game was an *NP*-complete problem.

He did this by showing that simple electrical circuit patterns with logical gates for *AND*, *OR*, *NOT* could be replicated by patterns of numbers on the board.
\[
\begin{array}{c}
\text{X} \rightarrow \\
\vdots \quad 1 \quad 1 \quad 1 \quad 2 \quad 2 \quad 1 \\
\vdots \quad x' \quad x \quad 2 \quad x' \quad * \quad * \quad 2 \\
\vdots \quad 1 \quad 1 \quad 1 \quad 1 \quad 2 \quad x' \quad * \quad 2 \\
\vdots \quad 1 \quad x' \quad 1 \\
\quad 1 \quad x \quad 1 \quad \text{X} \\
\quad 1 \quad 1 \quad 1 \\
\vdots \quad \vdots \quad \vdots \\
\end{array}
\]
More information

- For general information, not mathematical: 

- A more mathematical description at the undergraduate level: 
  *Introduction to Computation*, M. Sipser, MIT Press.

- For minesweeper: Kaye’s webpages and articles: 
  http://web.mat.bham.ac.uk/R.W.Kaye/minesw/