Millenium Problem: P $\neq$ NP

David Laferrière

March 27, 2007

What is an algorithm?

What is an algorithm?

- a finite set of well-defined instructions designed to solve a type of problem

What is an algorithm?

- a finite set of well-defined instructions designed to solve a type of problem
- in general, receives *input* and provides results as a final state called *output*

What is an algorithm?

- a finite set of well-defined instructions designed to solve a type of problem
- in general, receives *input* and provides results as a final state called *output*
- must terminate after a finite number of instructions

Example

Example

```
procedure SumFirstN (n: positive integer) begin
  x:=0
  for i:=1 to n do
    x:=x+i
end
```

Example

```
procedure SumFirstN (n: positive integer) begin
  x:=0
  for i:=1 to n do
    x:=x+i
end
```

So SumFirstN(5) would output the integer 15.

How do we determine complexity?

How do we determine complexity?

- Is there a way to measure the time required to solve a problem of a given size?

How do we determine complexity?

- Is there a way to measure the time required to solve a problem of a given size?
- Given two or more algorithms that solve the same problem, can we decide which is faster?

How do we determine complexity?

- Is there a way to measure the time required to solve a problem of a given size?
- Given two or more algorithms that solve the same problem, can we decide which is faster?
- Of course, the answer to both question is "Yes"

Time-Complexity Function

Time-Complexity Function

The *time-complexity function* $f(n)$ is the function which tells us the time required by an algorithm to determine an output given an input of length n.

Time-Complexity Function

The *time-complexity function* $f(n)$ is the function which tells us the time required by an algorithm to determine an output given an input of length n.
But how do we relate the time-complexity functions of two different algorithms?

What is time?

What is time?

- We define $f(n)$ to be the number of operations required to determine an output given an input of length n

What is time?

- We define $f(n)$ to be the number of operations required to determine an output given an input of length n
- Of course, the exact number of steps depends on the computer or the language used

What is time?

- We define $f(n)$ to be the number of operations required to determine an output given an input of length n
- Of course, the exact number of steps depends on the computer or the language used
- Instead we use *big-oh* notation

Big-oh notation

Definition

Let $f, g : \mathbb{N} \to \mathbb{R}$. We say that g *dominates* f if there exist constants $m \in \mathbb{R}^+$ and $k \in \mathbb{N}$ such that $|f(n)| \leq m|g(n)|$ for all $n \in \mathbb{N}, n \geq k$.

When f is dominated by g, we denote this by $f(n) \in O(g(n))$.

History

History

- Computability of problems was first considered by Church, Turing and Gödel in the 1930's

History

- Computability of problems was first considered by Church, Turing and Gödel in the 1930's
- Polynomial-time computation was introduced in the 1960s by Cobham and Edmonds

History

- Computability of problems was first considered by Church, Turing and Gödel in the 1930's
- Polynomial-time computation was introduced in the 1960s by Cobham and Edmonds
- Cook gave the first proof that a problem was NP-complete in 1971

History

- Computability of problems was first considered by Church, Turing and Gödel in the 1930's
- Polynomial-time computation was introduced in the 1960s by Cobham and Edmonds
- Cook gave the first proof that a problem was NP-complete in 1971
- Karp in 1972 presented a collection of other NP-complete problems

What is a Problem?

What is a Problem?

A *decision problem* Π is a set D_Π of *instances* subject to a question and a subset $Y_\Pi \subseteq D_\Pi$ of *yes-instances*.

P

P

- P is the class of all decision problems which can be solved by a deterministic Turing machine (DTM) in polynomial time

P

- P is the class of all decision problems which can be solved by a deterministic Turing machine (DTM) in polynomial time
- P is the class of so-called "tractable" problems

NP

NP

- Stands for "**n**on-deterministic **p**olynomial time"

NP

- Stands for "**n**on-deterministic **p**olynomial time"
- NP is the complexity class of decision problems that can be solved by a non-deterministic Turing machine (NDTM) in polynomial time

NP

- Stands for "**n**on-deterministic **p**olynomial time"
- NP is the complexity class of decision problems that can be solved by a non-deterministic Turing machine (NDTM) in polynomial time
- NP is equivalently the class of decision problems whose solutions can be verified in polynomial time

Non-deterministic Algorithm

Non-deterministic Algorithm

- Composed of two separate stages

Non-deterministic Algorithm

- Composed of two separate stages
- Guessing Stage

Non-deterministic Algorithm

- Composed of two separate stages
- Guessing Stage
  - Given a problem, the algorithm guesses a solution

Non-deterministic Algorithm

- Composed of two separate stages
- Guessing Stage
  - Given a problem, the algorithm guesses a solution
- Checking Stage

Non-deterministic Algorithm

- Composed of two separate stages
- Guessing Stage
  - Given a problem, the algorithm guesses a solution
- Checking Stage
  - Given the problem and the guess as inputs, checking stage computes deterministically

Non-deterministic Algorithm

- Composed of two separate stages
- Guessing Stage
  - Given a problem, the algorithm guesses a solution
- Checking Stage
  - Given the problem and the guess as inputs, checking stage computes deterministically
  - Returns "yes," returns "no," or does not halt

Reducibility

Reducibility

Let Π_1 and Π_2 be problems.

Reducibility

Let Π_1 and Π_2 be problems. We say that Π_1 *reduces to* Π_2 if there is a deterministic algorithm which transforms instances $d \in D_{\Pi_1}$ into instances in $c \in D_{\Pi_2}$, such that the answer to d is YES iff the answer to c is YES.

Reducibility

Let Π_1 and Π_2 be problems. We say that Π_1 *reduces to* Π_2 if there is a deterministic algorithm which transforms instances $d \in D_{\Pi_1}$ into instances in $c \in D_{\Pi_2}$, such that the answer to d is YES iff the answer to c is YES.
Denoted $\Pi_1 \propto \Pi_2$.

NP-Complete

NP-Complete

A decision problem is *NP-Hard* if every problem in *NP* is reducible to it in polynomial time.

NP-Complete

A decision problem is *NP-Hard* if every problem in *NP* is reducible to it in polynomial time.
A decision problem is *NP-Complete* if it is in *NP* and it is NP-Hard.

Satisfiability Problem

Satisfiability Problem

- $U = \{u_1, u_2, \ldots, u_m\}$ a set of Boolean variables

Satisfiability Problem

- $U = \{u_1, u_2, \ldots, u_m\}$ a set of Boolean variables
- A *truth assignment* $t : U \to \{T, F\}$

Satisfiability Problem

- $U = \{u_1, u_2, \ldots, u_m\}$ a set of Boolean variables
- A *truth assignment* $t : U \rightarrow \{T, F\}$
- If $u \in U$, u and $\overline{u}$ are *literals* over U

Satisfiability Problem

- $U = \{u_1, u_2, \ldots, u_m\}$ a set of Boolean variables
- A *truth assignment* $t : U \rightarrow \{T, F\}$
- If $u \in U$, u and $\overline{u}$ are *literals* over U
- A *clause* over U is a set of literals over U

Satisfiability Problem

- $U = \{u_1, u_2, \ldots, u_m\}$ a set of Boolean variables
- A *truth assignment* $t : U \to \{T, F\}$
- If $u \in U$, u and $\overline{u}$ are *literals* over U
- A *clause* over U is a set of literals over U
- A clause is *satisfied* by a truth assignment iff at least one of its members is true under that assignment

Satisfiability Problem

- $U = \{u_1, u_2, \ldots, u_m\}$ a set of Boolean variables
- A *truth assignment* $t : U \to \{T, F\}$
- If $u \in U$, u and $\overline{u}$ are *literals* over U
- A *clause* over U is a set of literals over U
- A clause is *satisfied* by a truth assignment iff at least one of its members is true under that assignment
- A collection C of clauses over U is *satisfiable* iff there exists some truth assignment for U that simultaneously satisfies each of the clauses in C

Example

Example

Let $U = \{u_1, u_2, u_3\}$, and let $t(u_1) = T$, $t(u_2) = T$, and let $t(u_3) = F$. Then $\{u_1, u_3\}$ is satisfied by t.

Example

Let $U = \{u_1, u_2, u_3\}$, and let $t(u_1) = T$, $t(u_2) = T$, and let $t(u_3) = F$. Then $\{u_1, u_3\}$ is satisfied by t.
$C = \{\{u_1, \overline{u}_2\}, \{\overline{u}_1, u_2\}\}$ is satisfiable.

Example

Let $U = \{u_1, u_2, u_3\}$, and let $t(u_1) = T$, $t(u_2) = T$, and let $t(u_3) = F$. Then $\{u_1, u_3\}$ is satisfied by t.
$C = \{\{u_1, \overline{u}_2\}, \{\overline{u}_1, u_2\}\}$ is satisfiable.
$C' = \{\{u_1, u_2\}, \{u_1, \overline{u}_2\}, \{\overline{u}_1\}\}$ is *not* satisfiable.

Satisfiability Problem

Satisfiability Problem

INSTANCE:

Satisfiability Problem

INSTANCE:

QUESTION:

Satisfiability Problem

INSTANCE: A set U of variables and a collection C of clauses over U.
QUESTION:

Satisfiability Problem

INSTANCE: A set U of variables and a collection C of clauses over U.
QUESTION: Is there a satisfying truth assignment for C?

Cook's Theorem

Cook's Theorem

Theorem (Cook, 1971)

The Satisfiability Problem is NP-Complete.

Cook's Theorem

Theorem (Cook, 1971)

The Satisfiability Problem is NP-Complete.

Proof.

It is easy to see that satisfiability is in NP.

Cook's Theorem

Theorem (Cook, 1971)

The Satisfiability Problem is NP-Complete.

Proof.

It is easy to see that satisfiability is in NP. The other direction is much more difficult (six pages in Garey and Johnson). $\square$

Traveling Salesman Problem (TSP)

Traveling Salesman Problem (TSP)

Instance:

Traveling Salesman Problem (TSP)

Instance: A finite set $C = \{c_1, c_2, \ldots, c_m\}$ of cities, a distance $d(c_i, c_j) \in \mathbb{Z}^+$ for each pair of cities $c_i, c_j \in C$, and a bound $B \in \mathbb{Z}^+$.

Traveling Salesman Problem (TSP)

Instance: A finite set $C = \{c_1, c_2, \ldots, c_m\}$ of cities, a distance $d(c_i, c_j) \in \mathbb{Z}^+$ for each pair of cities $c_i, c_j \in C$, and a bound $B \in \mathbb{Z}^+$.
Question:

Traveling Salesman Problem (TSP)

Instance: A finite set $C = \{c_1, c_2, \ldots, c_m\}$ of cities, a distance $d(c_i, c_j) \in \mathbb{Z}^+$ for each pair of cities $c_i, c_j \in C$, and a bound $B \in \mathbb{Z}^+$.
Question: Is there a tour of all cities in C having total length no more than B?

Hamiltonian Circuit (HC)

Hamiltonian Circuit (HC)

Instance:

Hamiltonian Circuit (HC)

Instance: A graph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_m\}$.

Hamiltonian Circuit (HC)

Instance: A graph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_m\}$.
Question:

Hamiltonian Circuit (HC)

Instance: A graph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_m\}$.
Question: Does G contain a Hamiltonian circuit?

Traveling Salesman vs Hamiltonian Circuit

Traveling Salesman vs Hamiltonian Circuit

Theorem

$HC \propto TS$

Traveling Salesman vs Hamiltonian Circuit

Theorem

$HC \propto TS$

Proof.

First we require a function f that maps each instance of HC to a corresponding instance of TS that satisfies the two properties required of a polynomial reduction.

Traveling Salesman vs Hamiltonian Circuit

Theorem

$HC \propto TS$

Proof.

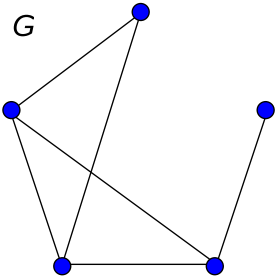
First we require a function f that maps each instance of HC to a corresponding instance of TS that satisfies the two properties required of a polynomial reduction.

Define f as follows: suppose $G = (V, E)$, with $|V| = m$, is an instance of HC. The corresponding instance of TS has a set C of cities that is identical to V. For any two cities $v_i, v_j \in C$, we define the inter-city distance $d(v_i, v_j)$ as follows:

$$d(v_i, v_j) = \begin{cases} 1 & \text{if } \{v_i, v_j\} \in E \\ 2 & \text{otherwise.} \end{cases}$$

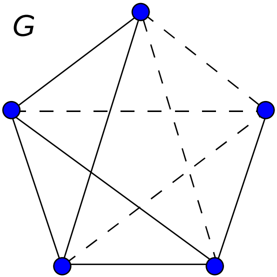
Example

Example



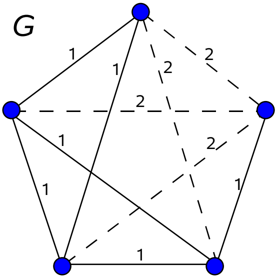
- Begin with a graph on five vertices.

Example



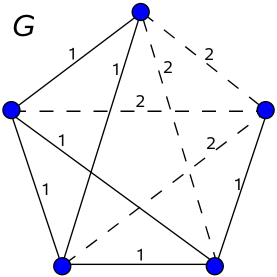
- Begin with a graph on five vertices.
- Make HC instance into TSP.

Example



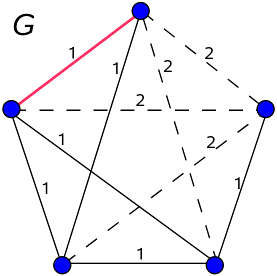
- Begin with a graph on five vertices.
- Make HC instance into TSP.
- Weight each edge.

Example



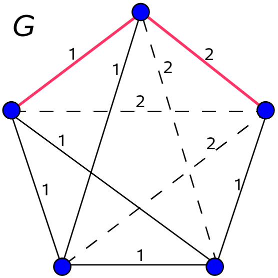
- Try to find a cycle of weight ≤ 5.

Example



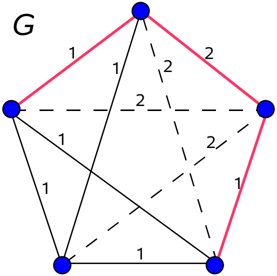
- Try to find a cycle of weight ≤ 5.

Example



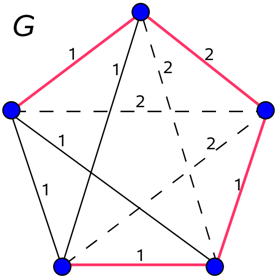
- Try to find a cycle of weight ≤ 5.

Example



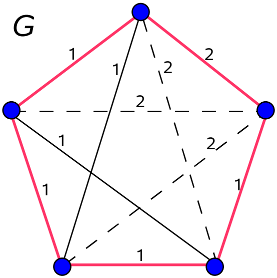
- Try to find a cycle of weight ≤ 5.

Example



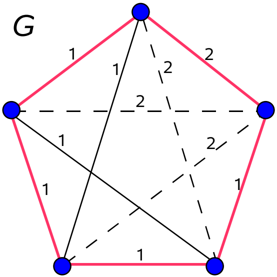
- Try to find a cycle of weight ≤ 5.

Example



- Try to find a cycle of weight ≤ 5.

Example



- Try to find a cycle of weight ≤ 5.
- None exist.

Traveling Salesman vs Hamiltonian Circuit

Proof (cont'd).

Traveling Salesman vs Hamiltonian Circuit

Proof (cont'd).
It is easy to see that f can be computed by a polynomial time algorithm.

Traveling Salesman vs Hamiltonian Circuit

Proof (cont'd).
It is easy to see that f can be computed by a polynomial time algorithm.
If we have a HC, then we have a tour of weight m.

Traveling Salesman vs Hamiltonian Circuit

Proof (cont'd).

It is easy to see that f can be computed by a polynomial time algorithm.

If we have a HC, then we have a tour of weight m.

If we have a tour of weight m, we must have a HC.

$\square$

Millenium Problem

Millenium Problem

Does $P = NP$?

Possible Answers

Possible Answers

- $P \neq NP$

Possible Answers

- $P \neq NP$
- $P = NP$ via existence proof

Possible Answers

- $P \neq NP$
- $P = NP$ via existence proof
- $P = NP$ via constructive proof

Consequences of $P \neq NP$

Consequences of $P \neq NP$

- Cryptography is safe for now

Consequences of $P \neq NP$

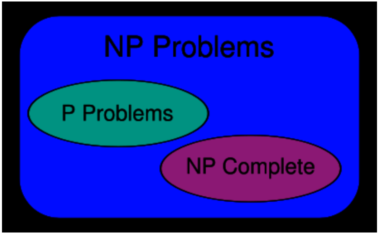
- Cryptography is safe for now
- The world of computational complexity makes sense

Consequences of $P \neq NP$

- Cryptography is safe for now
- The world of computational complexity makes sense
- There must exist problems in $NP - (P \bigcup NPC)$ [Ladner, 1975]

Consequences of $P \neq NP$

- Cryptography is safe for now
- The world of computational complexity makes sense
- There must exist problems in $NP - (P \bigcup NPC)$ [Ladner, 1975]



Consequences of $P = NP$ (Existence)

Consequences of $P = NP$ (Existence)

- Interesting result

Consequences of $P = NP$ (Existence)

- Interesting result
- Millenium Prize

Consequences of $P = NP$ (Existence)

- Interesting result
- Millenium Prize
- Hunt continues for constructive proof

Consequences of $P = NP$ (Constructively)

Consequences of $P = NP$ (Constructively)

- Depends on practicality of algorithm for reducing NP-complete problems to problems in P

Consequences of $P = NP$ (Constructively)

- Depends on practicality of algorithm for reducing NP-complete problems to problems in P
- If we are able to solve Satisfiability problem in (n^2) steps, we can factor a 200 digit number in minutes (devastating DES cryptography)

Consequences of $P = NP$ (Constructively)

- Depends on practicality of algorithm for reducing NP-complete problems to problems in P
- If we are able to solve Satisfiability problem in (n^2) steps, we can factor a 200 digit number in minutes (devastating DES cryptography)
- Practical algorithm for solving NP-complete problems would fundamentally change mathematics

Consequences of $P = NP$ (Constructively)

- Depends on practicality of algorithm for reducing NP-complete problems to problems in P
- If we are able to solve Satisfiability problem in (n^2) steps, we can factor a 200 digit number in minutes (devastating DES cryptography)
- Practical algorithm for solving NP-complete problems would fundamentally change mathematics
  - Rather than trying to prove results, we would devote our energies to *finding* interesting problems

Consequences of $P = NP$ (Constructively)

- Depends on practicality of algorithm for reducing NP-complete problems to problems in P
- If we are able to solve Satisfiability problem in (n^2) steps, we can factor a 200 digit number in minutes (devastating DES cryptography)
- Practical algorithm for solving NP-complete problems would fundamentally change mathematics
  - Rather than trying to prove results, we would devote our energies to *finding* interesting problems
  - We could potentially solve *all* the Millenium Problems

Current Status

Current Status

- Devlin calls the P versus NP problem "the one most likely to be solved by an unknown amateur."

Current Status

- Devlin calls the P versus NP problem "the one most likely to be solved by an unknown amateur."
- The more we learn, the further we seem from a solution

Current Status

- Devlin calls the P versus NP problem "the one most likely to be solved by an unknown amateur."
- The more we learn, the further we seem from a solution
- Razborov and Rudich (1993) showed that, given a certain credible assumption, "natural" proofs cannot solve $P = NP$

Current Status

- Devlin calls the P versus NP problem "the one most likely to be solved by an unknown amateur."
- The more we learn, the further we seem from a solution
- Razborov and Rudich (1993) showed that, given a certain credible assumption, "natural" proofs cannot solve $P = NP$
- Potential solution is a polynomial-time algorithm which solves an NP-complete problem

Thank You.