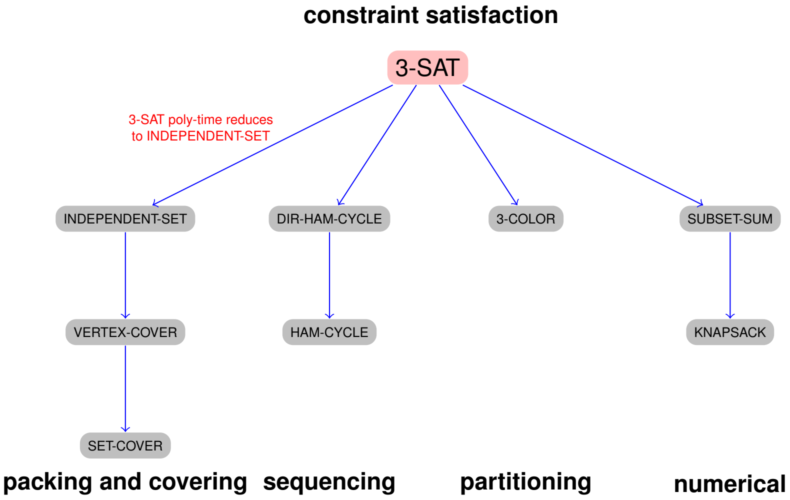
Design and Analysis of Algorithms XV

Complexity Classes

Guoqiang Li
School of Software

SHANGHAI JIAO TONG UNIVERSITY


constraint satisfaction
3-SAT
3-SAT poly-time reduces
to INDEPENDENT-SET
INDEPENDENT-SET
DIR-HAM-CYCLE
3-COLOR
SUBSET-SUM
VERTEX-COVER
HAM-CYCLE
KNAPSACK
SET-COVER
packing and covering
sequencing
partitioning
numerical

P VS. NP

Decision problem.

- Problem X is a set of strings.
- Instance s is one string.
- Algorithm A solves problem X: $A(s) = \begin{cases} yes & \text{if } s \in X \\ no & \text{if } s \notin X \end{cases}$

Decision problem.

- Problem X is a set of strings.
- Instance s is one string.
- Algorithm A solves problem X: $A(s) = \begin{cases} yes & \text{if } s \in X \\ no & \text{if } s \notin X \end{cases}$

Algorithm A runs in polynomial time if for every string s, $A(s)$ terminates in $\leq p(|s|)$ "steps", where $p(\cdot)$ is some polynomial function.

Decision problem.

- Problem X is a set of strings.
- Instance s is one string.
- Algorithm A solves problem X: $A(s) = \begin{cases} yes & \text{if } s \in X \\ no & \text{if } s \notin X \end{cases}$

Algorithm A runs in polynomial time if for every string s, $A(s)$ terminates in $\leq p(|s|)$ "steps", where $p(\cdot)$ is some polynomial function.

$\mathbf{P}$: set of decision problems for which there exists a poly-time algorithm.

Decision problem.

- Problem X is a set of strings.
- Instance s is one string.
- Algorithm A solves problem X: $A(s) = \begin{cases} yes & \text{if } s \in X \\ no & \text{if } s \notin X \end{cases}$

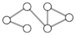
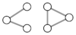
Algorithm A runs in polynomial time if for every string s, $A(s)$ terminates in $\leq p(|s|)$ "steps", where $p(\cdot)$ is some polynomial function.

$\mathbf{P}$: set of decision problems for which there exists a poly-time algorithm.

problem PRIMES**:**	$\{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, \ldots\}$
instance s**:**	592335744548702854681
algorithm:	Agrawal-Kayal-Saxena (2002)

Some problems in P

P. Decision problems for which there exists a poly-time algorithm.

problem	description	poly-time algorithm	yes	no
MULTIPLE	Is x a multiple of y?	grade-school division	51, 17	51, 16
REL-PRIME	Are x and y relatively prime?	Euclid's algorithm	34, 39	34, 51
PRIMES	Is x prime?	Agrawal-Kayal-Saxena	53	51
EDIT-DISTANCE	Is the edit distance between x and y less than 5?	Needleman–Wunsch	niether neither	acgggt ttttta
L-SOLVE	Is there a vector x that satisfies $Ax = b$?	Gauss–Edmonds elimination	$\begin{bmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & 15 \end{bmatrix}, \begin{bmatrix} 4 \\ 2 \\ 36 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$
U-CONN	Is an undirected graph G connected?	depth-first search		

NP

Definition. Algorithm $C(s,t)$ is a certifier for problem X if for every string s:$s \in X$ iff there exists a string t such that $C(s,t) = yes$.

NP

Shanghai Jiao Tong University

Definition. Algorithm $C(s,t)$ is a certifier for problem X if for every string s:$s \in X$ iff there exists a string t such that $C(s,t) = yes$.

NP: set of decision problems for which there exists a poly-time certifier.

- $C(s,t)$ is a poly-time algorithm.
- Certificate t is of polynomial size: $|t| \leq p(|s|)$ for some polynomial $p(\cdot)$.

NP

Shanghai Jiao Tong University

Definition. Algorithm $C(s,t)$ is a certifier for problem X if for every string s:$s \in X$ iff there exists a string t such that $C(s,t) = yes$.

NP: set of decision problems for which there exists a poly-time certifier.

- $C(s,t)$ is a poly-time algorithm.
- Certificate t is of polynomial size: $|t| \leq p(|s|)$ for some polynomial $p(\cdot)$.

problem COMPOSITES:	$\{4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, \ldots\}$
instance s:	437669
certificate t:	$541 \longleftarrow 437,669 = 541 \times 809$
certifier $\mathrm{C(s,t)}$:	grade school division

Certifiers and certificates: satisfiability

SAT. Given a CNF formula Φ, does it have a satisfying truth assignment?

3-SAT. SAT where each clause contains exactly 3 literals.

Certifiers and certificates: satisfiability

SAT. Given a CNF formula Φ, does it have a satisfying truth assignment?

3-SAT. SAT where each clause contains exactly 3 literals.

Certificate. An assignment of truth values to the Boolean variables.

Certifiers and certificates: satisfiability

SAT. Given a CNF formula Φ, does it have a satisfying truth assignment?

3-SAT. SAT where each clause contains exactly 3 literals.

Certificate. An assignment of truth values to the Boolean variables.

Certifier. Check that each clause in Φ has at least one true literal.

instance s $\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$

certificate t $x_1 =$ true , $x_2 =$ true , $x_3 =$ false , $x_4 =$ false

Certifiers and certificates: satisfiability

SAT. Given a CNF formula Φ, does it have a satisfying truth assignment?

3-SAT. SAT where each clause contains exactly 3 literals.

Certificate. An assignment of truth values to the Boolean variables.

Certifier. Check that each clause in Φ has at least one true literal.

instance s $\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$

certificate t $x_1 =$ true , $x_2 =$ true , $x_3 =$ false , $x_4 =$ false

Conclusions. SAT$\in$ **NP**, 3-SAT $\in$ **NP**

Certifiers and certificates: Hamilton path

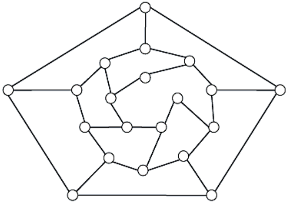
Hamilton Path. Given an undirected graph $G = (V, E)$, does there exist a simple path P that visits every node?

Certifiers and certificates: Hamilton path

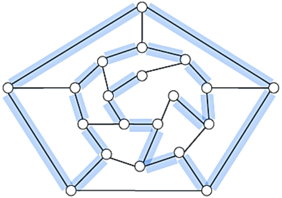
Hamilton Path. Given an undirected graph $G = (V, E)$, does there exist a simple path P that visits every node?

Certificate. A permutation π of the n nodes.

Certifier. Check that π contains each node in V exactly once, and that G contains an edge between each pair of adjacent nodes.



instance s



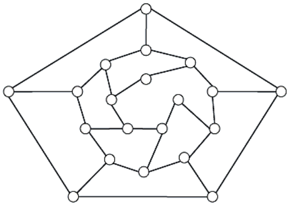
certificate t

Certifiers and certificates: Hamilton path

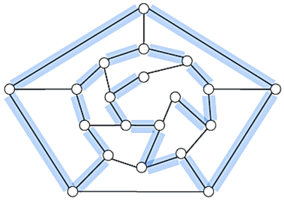
Hamilton Path. Given an undirected graph $G = (V, E)$, does there exist a simple path P that visits every node?

Certificate. A permutation π of the n nodes.

Certifier. Check that π contains each node in V exactly once, and that G contains an edge between each pair of adjacent nodes.



instance s

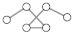
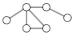


certificate t

Conclusion. Hamilton path $\in$ **NP.**

Some problems in NP

NP. Decision problems for which there exists a poly-time certifier.

problem	description	poly-time algorithm	yes	no
L-solve	Is there a vector x that satisfies $Ax = b$?	Gauss–Edmonds elimination	$\begin{bmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & 15 \end{bmatrix}, \begin{bmatrix} 4 \\ 2 \\ 36 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$
Composites	Is x composite ?	Agrawal-Kayal-Saxena	51	53
Factor	Does x have a nontrivial factor less than y ?	???	(56159, 50)	(55687, 50)
SAT	Given a CNF formula, does it have a satisfying truth assignment?	???	$\neg x_1 \vee x_2 \vee \neg x_3$ $x_1 \vee \neg x_2 \vee x_3$ $\neg x_1 \vee \neg x_2 \vee x_3$	$\neg x_2$ $x_1 \vee x_2$ $\neg x_1 \vee x_2$
Hamilton path	Is there a simple path between u and v that visits every node?	???		

Quiz

Which of the following graph problems are known to be in NP?

- A. Is the length of the longest simple path $\leq k$?
- B. Is the length of the longest simple path $\geq k$?
- C. Is the length of the longest simple path $= k$?
- D. Find the length of the longest simple path.
- E. All of the above.

Quiz

In complexity theory, the abbreviation NP stands for . . .

A. Nope.
B. No problem.
C. Not polynomial time.
D. Not polynomial space.
E. Nondeterministic polynomial time.

Significance of NP

NP. Decision problems for which there exists a poly-time certifier.

Significance of NP

NP. Decision problems for which there exists a poly-time certifier.

> *"NP captures vast domains of computational, scientific, and mathematical endeavors, and seems to roughly delimit what mathematicians and scientists have been aspiring to compute feasibly."*
>
> *–Christos Papadimitriou*

Significance of NP

NP. Decision problems for which there exists a poly-time certifier.

"NP captures vast domains of computational, scientific, and mathematical endeavors, and seems to roughly delimit what mathematicians and scientists have been aspiring to compute feasibly."

–Christos Papadimitriou

"In an ideal world it would be renamed P vs VP. "

–Clyde Kruskal

P, NP, and EXP

P. Decision problems for which there exists a poly-time algorithm.

NP. Decision problems for which there exists a poly-time certifier.

EXP. Decision problems for which there exists an exponential-time algorithm.

P, NP, and EXP

Proposition. **P** $\subseteq$ **NP**.

P, NP, and EXP

Proposition. **P** $\subseteq$ **NP**.

Proof.

P, NP, and EXP

Proposition. **P** $\subseteq$ **NP**.

Proof. Consider any problem $X \in$ **P**.

- By definition, there exists a poly-time algorithm $A(s)$ that solves X.
- Certificate $t = \varepsilon$, certifier $C(s,t) = A(s)$.

P, NP, and EXP

Shanghai Jiao Tong University

Proposition. **P** $\subseteq$ **NP**.

Proof. Consider any problem $X \in$ **P**.

- By definition, there exists a poly-time algorithm $A(s)$ that solves X.
- Certificate $t = \varepsilon$, certifier $C(s, t) = A(s)$.

Proposition. **NP** $\subseteq$ **EXP**.

P, NP, and EXP

Proposition. **P** $\subseteq$ **NP**.

Proof. Consider any problem $X \in$ **P**.

- By definition, there exists a poly-time algorithm $A(s)$ that solves X.
- Certificate $t = \varepsilon$, certifier $C(s,t) = A(s)$.

Proposition. **NP** $\subseteq$ **EXP**.

Proof. Consider any problem $X \in$ **NP**.

P, NP, and EXP

Proposition. **P** $\subseteq$ **NP**.

Proof. Consider any problem $X \in$ **P**.

- By definition, there exists a poly-time algorithm $A(s)$ that solves X.
- Certificate $t = \varepsilon$, certifier $C(s,t) = A(s)$.

Proposition. **NP** $\subseteq$ **EXP**.

Proof. Consider any problem $X \in$ **NP**.

- By definition, there exists a poly-time certifier $C(s,t)$ for X, where certificate t satisfies $|t| \le p(|s|)$ for some polynomial $p(\cdot)$.
- To solve instance s, run $C(s,t)$ on all strings t with $|t| \le p(|s|)$.
- Return yes iff $C(s,t)$ returns yes for any of these potential certificates.

P, NP, and EXP

Proposition. **P** $\subseteq$ **NP**.

Proof. Consider any problem $X \in$ **P**.

- By definition, there exists a poly-time algorithm $A(s)$ that solves X.
- Certificate $t = \varepsilon$, certifier $C(s, t) = A(s)$.

Proposition. **NP** $\subseteq$ **EXP**.

Proof. Consider any problem $X \in$ **NP**.

- By definition, there exists a poly-time certifier $C(s, t)$ for X, where certificate t satisfies $|t| \leq p(|s|)$ for some polynomial $p(\cdot)$.
- To solve instance s, run $C(s, t)$ on all strings t with $|t| \leq p(|s|)$.
- Return yes iff $C(s, t)$ returns yes for any of these potential certificates.

Fact. **P** $\neq$ **EXP** $\Rightarrow$ either **P** $\neq$ **NP**, or **NP** $\neq$ **EXP**, or both.

The main question: P vs. NP

Q. How to solve an instance of 3-SAT with n variables?

The main question: P vs. NP

Q. How to solve an instance of 3-SAT with n variables?

A. Exhaustive search: try all 2^n truth assignments.

The main question: P vs. NP

Q. How to solve an instance of 3-SAT with n variables?

A. Exhaustive search: try all 2^n truth assignments.

Q. Can we do anything substantially more clever?

The main question: P vs. NP

SHANGHAI JIAO TONG UNIVERSITY

Q. How to solve an instance of 3-SAT with n variables?

A. Exhaustive search: try all 2^n truth assignments.

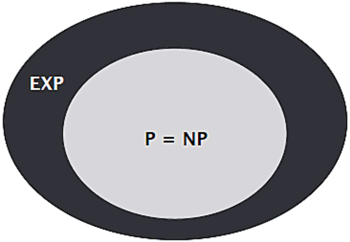
Q. Can we do anything substantially more clever?

Conjecture. No poly-time algorithm for 3-SAT.

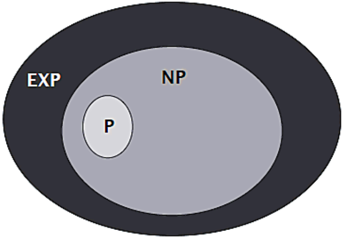
"intractable"

The main question: P vs. NP

Does **P** = **NP**? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]
Is the decision problem as easy as the certification problem?



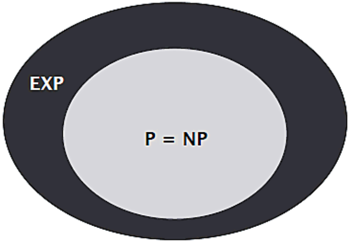
if P = NP



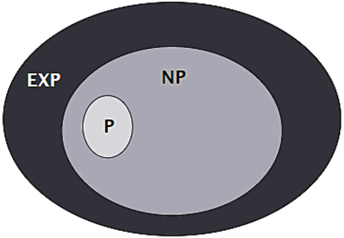
if P $\neq$ NP

The main question: P vs. NP

Does **P** = **NP**? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]
Is the decision problem as easy as the certification problem?



if P = NP



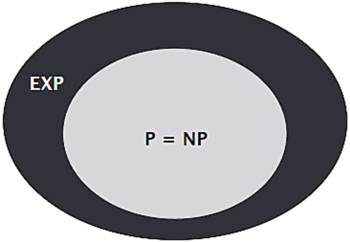
if P $\neq$ NP

If yes. . . Efficient algorithms for 3-SAT, TSP, VERTEX-COVER, FACTOR. . .

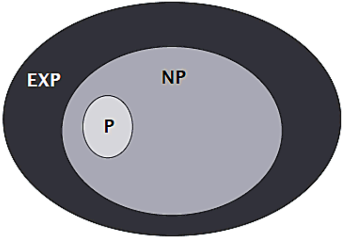
If no. . . No efficient algorithms possible for 3-SAT, TSP, VERTEX-COVER. . .

The main question: P vs. NP

Does **P** = **NP**? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]
Is the decision problem as easy as the certification problem?



if P = NP



if P $\neq$ NP

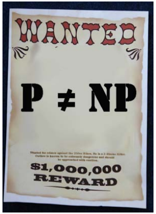
If yes. . . Efficient algorithms for 3-SAT, TSP, VERTEX-COVER, FACTOR. . .

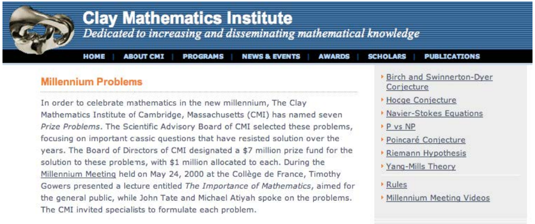
If no. . . No efficient algorithms possible for 3-SAT, TSP, VERTEX-COVER. . .

Consensus opinion. Probably no.

Millennium prize

Millennium prize. \$1 million for resolution of $\mathbf{P} \neq \mathbf{NP}$ problem.





NP-complete

Polynomial transformations

SHANGHAI JIAO TONG UNIVERSITY

Definition

Problem X polynomial (Cook) reduces to problem Y if arbitrary instances of problem X can be solved using:

- polynomial number of standard computational steps, and
- Polynomial number of calls to oracle that solves problem Y.

Polynomial transformations

Definition

Problem X polynomial (Cook) reduces to problem Y if arbitrary instances of problem X can be solved using:

- polynomial number of standard computational steps, and
- Polynomial number of calls to oracle that solves problem Y.

Definition

Problem X polynomial (Karp) transforms to problem Y if given any instance x of X, we can construct an instance y of Y such that x is a yes instance of X iff y is a yes instance of Y.

Polynomial transformations

Definition

Problem X polynomial (Cook) reduces to problem Y if arbitrary instances of problem X can be solved using:

- polynomial number of standard computational steps, and
- Polynomial number of calls to oracle that solves problem Y.

Definition

Problem X polynomial (Karp) transforms to problem Y if given any instance x of X, we can construct an instance y of Y such that x is a yes instance of X iff y is a yes instance of Y.

Note. Polynomial transformation is polynomial reduction with just one call to oracle for Y, exactly at the end of the algorithm for X. Almost all previous reductions were of this form.

Polynomial transformations

SHANGHAI JIAO TONG UNIVERSITY

Definition

Problem X polynomial (Cook) reduces to problem Y if arbitrary instances of problem X can be solved using:

- polynomial number of standard computational steps, and
- Polynomial number of calls to oracle that solves problem Y.

Definition

Problem X polynomial (Karp) transforms to problem Y if given any instance x of X, we can construct an instance y of Y such that x is a yes instance of X iff y is a yes instance of Y.

Note. Polynomial transformation is polynomial reduction with just one call to oracle for Y, exactly at the end of the algorithm for X. Almost all previous reductions were of this form.

Open question. Are these two concepts the same with respect to **NP**?

NP-complete

NP-complete. A problem $Y \in$ **NP** with the property that for every problem $X \in$ **NP**, $X \leq_P Y$.

NP-complete

NP-complete. A problem $Y \in$ **NP** with the property that for every problem $X \in$ **NP**, $X \leq_P Y$.

Proposition

*Suppose $Y \in$ **NP**-complete. Then, $Y \in$ **P** iff **P** $=$ **NP**.*

NP-complete. A problem $Y \in$ **NP** with the property that for every problem $X \in$ **NP**, $X \leq_P Y$.

Proposition

*Suppose $Y \in$ **NP**-complete. Then, $Y \in$ **P** iff **P** $=$ **NP**.*

Proof.

NP-complete

NP-complete. A problem $Y \in$ **NP** with the property that for every problem $X \in$ **NP**, $X \leq_P Y$.

Proposition

*Suppose $Y \in$ **NP**-complete. Then, $Y \in$ **P** iff **P** $=$ **NP**.*

Proof.

$\Leftarrow$ If **P** $=$ **NP**, then $Y \in$ **P**.

NP-complete

NP-complete. A problem $Y \in$ **NP** with the property that for every problem $X \in$ **NP**, $X \leq_P Y$.

Proposition

*Suppose $Y \in$ **NP**-complete. Then, $Y \in$ **P** iff **P** $=$ **NP**.*

Proof.

$\Leftarrow$ If **P** $=$ **NP**, then $Y \in$ **P**.

$\Rightarrow$ Suppose $Y \in$ **P**.

- Consider any problem $X \in$ **NP**. Since $X \leq_P Y$, we have $X \in$ **P**.
- This implies **NP** $\subseteq$ **P**.
- We already know **P** $\subseteq$ **NP**. Thus **P**= **NP**.

NP-complete

NP-complete. A problem $Y \in$ **NP** with the property that for every problem $X \in$ **NP**, $X \leq_P Y$.

Proposition

*Suppose $Y \in$ **NP**-complete. Then, $Y \in$ **P** iff **P** $=$ **NP**.*

Proof.

$\Leftarrow$ If **P** $=$ **NP**, then $Y \in$ **P**.

$\Rightarrow$ Suppose $Y \in$ **P**.

- Consider any problem $X \in$ **NP**. Since $X \leq_P Y$, we have $X \in$ **P**.
- This implies **NP** $\subseteq$ **P**.
- We already know **P** $\subseteq$ **NP**. Thus **P**= **NP**.

Fundamental question. Are there any "natural" **NP**-complete problems?

The "first" NP-complete problem

Theorem (Cook 1971, Levin 1973)

SAT $\in$ ***NP***-*complete.*

The Complexity of Theorem-Proving Procedures

Stephen A. Cook

University of Toronto

Summary

It is shown that any recognition problem solved by a polynomial time-bounded nondeterministic Turing machine can be "reduced" to the problem of determining whether a given propositional formula is a tautology. Here "reduced" means, roughly speaking, that the first problem can be solved deterministically in polynomial time provided an oracle is available for solving the second. From this notion of reducible, polynomial degrees of difficulty are defined, and it is shown that the problem of determining tautologyhood has the same polynomial degree as the problem of determining whether the first of two given graphs is isomorphic to a subgraph of the second. Other examples are discussed. A method of measuring the complexity of proof procedures for the predicate calculus is introduced and discussed.

Throughout this paper, a set of strings means a set of strings on some fixed, large, finite alphabet Σ. This alphabet is large enough to include symbols for all sets described here. All Turing machines are deterministic recognition devices, unless the contrary is explicitly stated.

1. Tautologies and Polynomial Re-Reducibility.

Let us fix a formalism for the propositional calculus in which formulas are written as strings on Σ. Since we will require infinitely many proposition symbols (atoms), each such symbol will consist of a member of Σ followed by a number in binary notation to distinguish that symbol. Thus a formula of length n can only have about n/logn distinct function and predicate symbols. The logical connectives are & (and), ∨ (or), and ¬(not).

The set of tautologies (denoted by {tautologies}) is a certain recursive set of strings on this alphabet, and we are interested in the problem of finding a good lower bound on its possible recognition times. We provide no such lower bound here, but theorem 1 will give evidence that {tautologies} is a difficult set to recognize, since many apparently difficult problems can be reduced to determining tautologyhood. By reduced we mean, roughly speaking, that if tautologyhood could be decided instantly (by an "oracle") then these problems could be decided in polynomial time. In order to make this notion precise, we introduce query machines, which are like Turing machines with oracles in [1].

A query machine is a multitape Turing machine with a distinguished tape called the query tape, and three distinguished states called the query state, yes state, and no state, respectively. If M is a query machine and T is a set of strings, then a T-computation of M is a computation of M in which initially M is in the initial state and has an input string w on its input tape, and each time M assumes the query state there is a string u on the query tape, and the next state M assumes is the yes state if u∈T and the no state if u∉T. We think of an "oracle", which knows T, placing M in the yes state or no state.

Definition

A set S of strings is P-reducible (P for polynomial) to a set T of strings iff there is some query machine M and a polynomial Q(n) such that for each input string w, the T-computation of M with input w halts within Q(|w|) steps (|w| is the length of w), and ends in an accepting state iff w∈S.

It is not hard to see that P-reducibility is a transitive relation. Thus the relation E on

ПРОБЛЕМЫ ПЕРЕДАЧИ ИНФОРМАЦИИ

Том IX *1973* *Вып. 3*

КРАТКИЕ СООБЩЕНИЯ

УДК 519.14

УНИВЕРСАЛЬНЫЕ ЗАДАЧИ ПЕРЕБОРА

Л. А. Левин

В статье рассматривается несколько известных массовых задач «переборного типа» и доказывается, что эти задачи можно решать лишь за такое время, за которое можно решать вообще любые задачи указанного типа.

После уточнения понятия алгоритма была доказана алгоритмическая неразрешимость ряда классических массовых проблем (например, проблем тождества элементов групп, гомеоморфности многообразий, разрешимости диофантовых уравнений и других). Тем самым был снят вопрос о нахождении практического способа их решения. Однако существование алгоритмов для решения других задач не снимает для них аналогичного вопроса из-за фантастически большого объема работы, предписываемого этими алгоритмами. Такова ситуация с так называемыми переборными задачами: минимизации булевых функций, поиска доказательств ограниченной длины, выяснения изоморфности графов и другими. Все эти задачи решаются тривиальными алгоритмами, состоящими в переборе всех возможностей. Однако эти алгоритмы требуют экспоненциального времени работы и у математиков сложилось убеждение, что более простые алгоритмы для них невозможны. Был получен ряд серьезных аргументов в пользу его справедливости (см.[1, 2]), однако доказать это утверждение не удалось никому. (Например, до сих пор не доказано, что для нахождения математических доказательств нужно больше времени, чем для их проверки.)

Однако если предположить, что вообще существует какая-нибудь (хотя бы искусственно построенная) массовая задача переборного типа, неразрешимая простыми (в смысле объема вычислений) алгоритмами, то можно показать, что этим же свойством обладают и многие «классические» переборные задачи (в том числе задача минимизации, задача поиска доказательств и др.). В этом и состоят основные результаты статьи.

Функции $f(n)$ и $g(n)$ будем называть сравнимыми, если при некотором k

$$f(n) \leqslant (g(n)+2)^k \quad \text{и} \quad g(n) \leqslant (f(n)+2)^k.$$

Аналогично будем понимать термин «меньше или сравнимо».

Определение. Задачей переборного типа (или просто переборной задачей) будем называть задачу вида «по данному x найти какое-нибудь y длины, сравнимой с длиной x, такое, что выполняется $A(x, y)$», где $A(x, y)$ – какое-нибудь свойство, проверяемое алгоритмом, время работы которого сравнимо с длиной x. (Под алгоритмом здесь можно понимать, например, алгоритмы Колмогорова — Успенского или машины Тьюринга, или нормальные алгоритмы; x, y – двоичные слова). Квазипереборной задачей будем называть задачу выяснения, существует ли такое y.

Мы рассмотрим шесть задач этих типов. Рассматриваемые в них объекты кодируются естественным образом в виде двоичных слов. При этом выбор естественной кодировки не существен, так как все они дают сравнимые длины кодов.

Задача 1. Заданы списком конечное множество и покрытие его 500-элементными подмножествами. Найти подпокрытие заданной мощности (соответственно выяснить существует ли оно).

Задача 2. Таблично задана частичная булева функция. Найти заданного размера дизъюнктивную нормальную форму, реализующую эту функцию в области определения (соответственно выяснить существует ли она).

Задача 3. Выяснить, выводима или опровержима данная формула исчисления высказываний. (Или, что то же самое, равна ли константе данная булева формула.)

Задача 4. Даны два графа. Найти гомоморфизм одного на другой (выяснить его существование).

Задача 5. Даны два графа. Найти изоморфизм одного в другой (на его часть).

Задача 6. Рассматриваются матрицы из целых чисел от 1 до 100 и некоторое условие о том, какие числа в них могут соседствовать по вертикали и какие по горизонтали. Заданы числа на границе и требуется продолжить их на всю матрицу с соблюдением условия.

Establishing NP-completeness

Remark. Once we establish first “natural” NP-complete problem, others fall like dominoes.

Establishing NP-completeness

Remark. Once we establish first "natural" NP-complete problem, others fall like dominoes.

Recipe. To prove that $Y \in$ **NP**-complete:

Establishing NP-completeness

Remark. Once we establish first “natural” NP-complete problem, others fall like dominoes.

Recipe. To prove that $Y \in$ **NP**-complete:

- Step 1. Show that $Y \in$ **NP**.
- Step 2. Choose an **NP**-complete problem X.
- Step 3. Prove that $X \leq_P Y$.

Establishing NP-completeness

Remark. Once we establish first "natural" NP-complete problem, others fall like dominoes.

Recipe. To prove that $Y \in$ **NP**-complete:

- Step 1. Show that $Y \in$ **NP**.
- Step 2. Choose an **NP**-complete problem X.
- Step 3. Prove that $X \leq_P Y$.

Proposition

*If $X \in$ **NP**-complete, $Y \in$ **NP**, and $X \leq_P Y$, then $Y \in$ **NP**-complete.*

Establishing NP-completeness

Remark. Once we establish first "natural" NP-complete problem, others fall like dominoes.

Recipe. To prove that $Y \in$ **NP**-complete:

- Step 1. Show that $Y \in$ **NP**.
- Step 2. Choose an **NP**-complete problem X.
- Step 3. Prove that $X \leq_P Y$.

Proposition

*If $X \in$ **NP**-complete, $Y \in$**NP**, and $X \leq_P Y$, then $Y \in$**NP**-complete.*

Proof. Consider any problem $W \in$ **NP**. Then, both $W \leq_P X$ and $X \leq_P Y$.

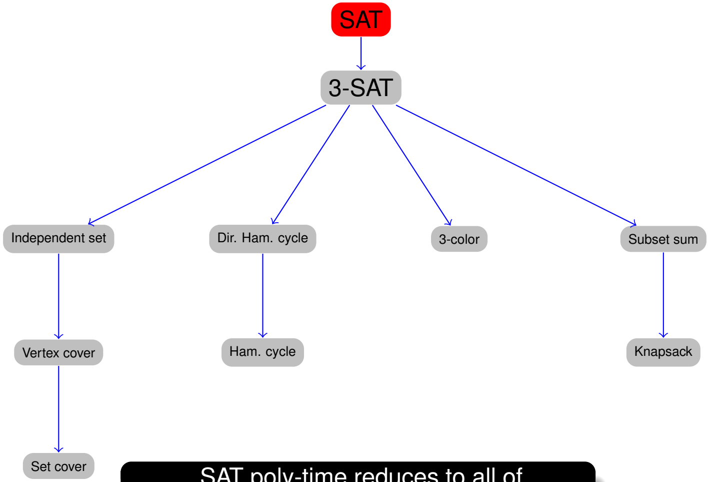
- By transitivity, $W \leq_P Y$.
- Hence $Y \in$ **NP**-complete.

Quiz

Suppose that $X \in$ **NP**-Complete, $Y \in$ **NP**, and $X \leq_P Y$. Which can you infer?

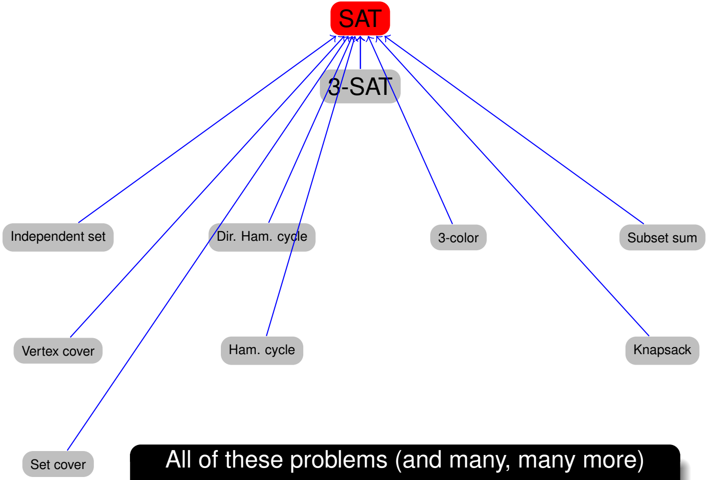
A. Y is **NP**-complete.
B. If $Y \notin$ **P**, then **P**$\neq$ **NP**.
C. If **P**$\neq$**NP**, then neither X nor Y is in **P**.
D. All of the above.

Implications of Karp



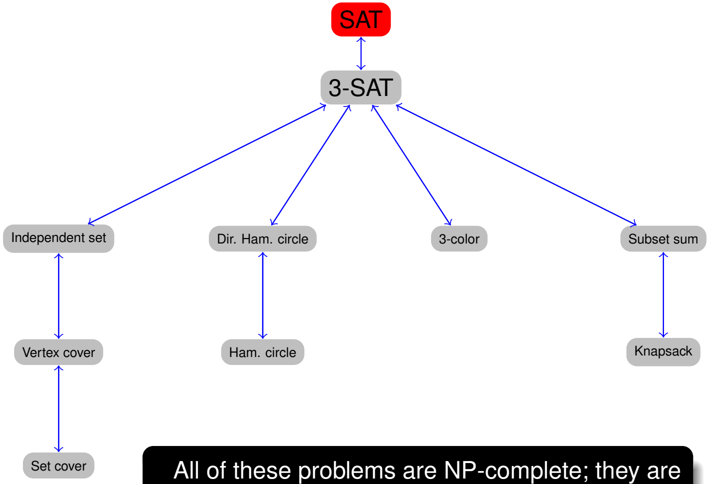
SAT poly-time reduces to all of these problems (and many, many more)

Implications of Cook–Levin



All of these problems (and many, many more) poly-time reduce to SAT .

Implications of Karp + Cook–Levin



All of these problems are NP-complete; they are manifestations of the same really hard problem.

Some NP-complete problems

Basic genres of **NP**-complete problems and paradigmatic examples.

Some NP-complete problems

Basic genres of **NP**-complete problems and paradigmatic examples.

- Packing/covering problems: Set cover, Vertex cover Independent set.

Some NP-complete problems

Basic genres of **NP**-complete problems and paradigmatic examples.

- Packing/covering problems: Set cover, Vertex cover Independent set.
- Constraint satisfaction problems: Circuit SAT, SAT, 3-SAT.

Some NP-complete problems

Basic genres of **NP**-complete problems and paradigmatic examples.

- Packing/covering problems: Set cover, Vertex cover Independent set.
- Constraint satisfaction problems: Circuit SAT, SAT, 3-SAT.
- Sequencing problems: Hamilton circle, TSP.

Some NP-complete problems

Basic genres of **NP**-complete problems and paradigmatic examples.

- Packing/covering problems: Set cover, Vertex cover Independent set.
- Constraint satisfaction problems: Circuit SAT, SAT, 3-SAT.
- Sequencing problems: Hamilton circle, TSP.
- Partitioning problems: 3D-matching, 3-color.

Some NP-complete problems

Shanghai Jiao Tong University

Basic genres of **NP**-complete problems and paradigmatic examples.

- Packing/covering problems: Set cover, Vertex cover Independent set.
- Constraint satisfaction problems: Circuit SAT, SAT, 3-SAT.
- Sequencing problems: Hamilton circle, TSP.
- Partitioning problems: 3D-matching, 3-color.
- Numerical problems: Subset sum, Knapsack.

Some NP-complete problems

Basic genres of **NP**-complete problems and paradigmatic examples.

- Packing/covering problems: Set cover, Vertex cover Independent set.
- Constraint satisfaction problems: Circuit SAT, SAT, 3-SAT.
- Sequencing problems: Hamilton circle, TSP.
- Partitioning problems: 3D-matching, 3-color.
- Numerical problems: Subset sum, Knapsack.

Practice. Most **NP** problems are known to be in either **P** or **NP**-complete.

Some NP-complete problems

Basic genres of **NP**-complete problems and paradigmatic examples.

- Packing/covering problems: Set cover, Vertex cover Independent set.
- Constraint satisfaction problems: Circuit SAT, SAT, 3-SAT.
- Sequencing problems: Hamilton circle, TSP.
- Partitioning problems: 3D-matching, 3-color.
- Numerical problems: Subset sum, Knapsack.

Practice. Most **NP** problems are known to be in either **P** or **NP**-complete.

NP-intermediate? Factor, Discrete log, Graph isomorphism, . . .

Some NP-complete problems

Basic genres of **NP**-complete problems and paradigmatic examples.

- Packing/covering problems: Set cover, Vertex cover Independent set.
- Constraint satisfaction problems: Circuit SAT, SAT, 3-SAT.
- Sequencing problems: Hamilton circle, TSP.
- Partitioning problems: 3D-matching, 3-color.
- Numerical problems: Subset sum, Knapsack.

Practice. Most **NP** problems are known to be in either **P** or **NP**-complete.

NP-intermediate? Factor, Discrete log, Graph isomorphism, . . .

Theorem (Ladner 1975)

*Unless **P = NP**, there exist problems in **NP** that are in neither **P** nor **NP**-complete.*

More hard computational problems

Shanghai Jiao Tong University

Garey and Johnson. *Computers and Intractability*.

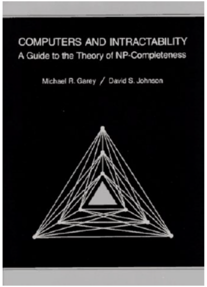
- Appendix includes over 300 **NP**-complete problems.
- Most cited reference in computer science literature.

Most Cited Computer Science Citations

This list is generated from documents in the CiteSeer^{X} database as of January 17, 2013. This list is automatically generated and may contain errors. The list is generated in batch mode and citation counts may differ from those currently in the CiteSeer^{X} database, since the database is continuously updated.

All Years | 1990 | 1991 | 1992 | 1993 | 1994 | 1995 | 1996 | 1997 | 1998 | 1999 | 2000 | 2001 | 2002 | 2003 | 2004 | 2005 | 2006 | 2007 | 2008 | 2009 | 2010 | 2011 | 2012 | 2013

1. M R Garey, D S Johnson
   Computers and Intractability. A Guide to the Theory of NP-Completeness 1979
   8665
2. T Cormen, C E Leiserson, R Rivest
   Introduction to Algorithms 1990
   7210
3. V N Vapnik
   The nature of statistical learning theory 1998
   6580
4. A P Dempster, N M Laird, D B Rubin
   Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society, 1977
   6082
5. T Cover, J Thomas
   Elements of Information Theory 1991
   6075
6. D E Goldberg
   Genetic Algorithms in Search, Optimization, and Machine Learning, 1989
   5998
7. J Pearl
   Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference 1988
   5582
8. E Gamma, R Helm, R Johnson, J Vlissides
   Design Patterns: Elements of Reusable Object-Oriented Software 1995
   4614
9. C E Shannon
   A mathematical theory of communication Bell Syst. Tech. J, 1948
   4118
10. J R Quinlan
    C4.5: Programs for Machine Learning 1993
    4018



More hard computational problems

SHANGHAI JIAO TONG UNIVERSITY

Aerospace engineering. Optimal mesh partitioning for finite elements.
Biology. Phylogeny reconstruction.
Chemical engineering. Heat exchanger network synthesis.
Chemistry. Protein folding.
Civil engineering. Equilibrium of urban traffic flow.
Economics. Computation of arbitrage in financial markets with friction.
Electrical engineering. VLSI layout.
Environmental engineering. Optimal placement of contaminant sensors.
Financial engineering. Minimum risk portfolio of given return.
Game theory. Nash equilibrium that maximizes social welfare.
Mathematics. Given integer $a_1, \ldots, a_n$, compute $\int_0^{2\pi} \cos(a_1\theta) \times \cos(a_2\theta) \times \cdots \times \cos(a_n\theta)\, d\theta$
Mechanical engineering. Structure of turbulence in sheared flows.
Medicine. Reconstructing 3d shape from biplane angiocardiogram.
Operations research. Traveling salesperson problem.
Physics. Partition function of 3d Ising model.
Politics. Shapley–Shubik voting power.
Recreation. Versions of Sudoku, Checkers, Minesweeper, Tetris, Rubik's Cube.
Statistics. Optimal experimental design.

co-NP

Asymmetry of NP

Asymmetry of NP. We need short certificates only for yes instances.

Asymmetry of NP

Asymmetry of NP. We need short certificates only for yes instances.

Example 1. SAT vs. Un-SAT.

- Can prove a CNF formula is satisfiable by specifying an assignment.
- How could we prove that a formula is not satisfiable?

Asymmetry of NP

Asymmetry of NP. We need short certificates only for yes instances.

Example 1. SAT vs. Un-SAT.

- Can prove a CNF formula is satisfiable by specifying an assignment.
- How could we prove that a formula is not satisfiable?

SAT. Given a CNF formula Φ, is there a satisfying truth assignment?

Un-SAT. Given a CNF formula Φ, is there no satisfying truth assignment?

Asymmetry of NP

Asymmetry of NP. We need short certificates only for yes instances.

Asymmetry of NP

Shanghai Jiao Tong University

Asymmetry of NP. We need short certificates only for yes instances.

Example 2. Hamilton cycle vs. No Hamilton cycle.

- Can prove a graph is Hamiltonian by specifying a permutation.
- How could we prove that a graph is not Hamiltonian?

Asymmetry of NP

Asymmetry of NP. We need short certificates only for yes instances.

Example 2. Hamilton cycle vs. No Hamilton cycle.

- Can prove a graph is Hamiltonian by specifying a permutation.
- How could we prove that a graph is not Hamiltonian?

HAMILTON CYCLE. Given a graph $G = (V, E)$, is there a simple cycle Γ that contains every node in V?

NO HAMILTON CYCLE. Given a graph $G = (V, E)$, is there no simple cycle Γ that contains every node in V?

Asymmetry of NP

Asymmetry of **NP**. We need short certificates only for yes instances.

Q. How to classify Un-SAT and No Hamilton cycle?

Asymmetry of NP

SHANGHAI JIAO TONG UNIVERSITY

Asymmetry of **NP**. We need short certificates only for yes instances.

Q. How to classify Un-SAT and No Hamilton cycle?

- SAT $\in$ **NP**-complete and SAT$\equiv_P$ Un-SAT.
- Hamilton circle $\in$ **NP**-complete and Hamilton circle $\equiv_P$ No Hamilton circle.
- But neither Un-SAT nor No Hamilton circle are known to be in **NP**.

NP and co-NP

SHANGHAI JIAO TONG UNIVERSITY

NP. Decision problems for which there is a poly-time certifier.
Example. SAT, Hamilton cycle, and Composites.

NP and co-NP

SHANGHAI JIAO TONG UNIVERSITY

NP. Decision problems for which there is a poly-time certifier.
Example. SAT, Hamilton cycle, and Composites.

Definition

Given a decision problem X, its complement $\overline{X}$ is the same problem with the yes and no answers reversed.

NP. Decision problems for which there is a poly-time certifier.
Example. SAT, Hamilton cycle, and Composites.

Definition

Given a decision problem X, its complement $\overline{X}$ is the same problem with the yes and no answers reversed.

Example $X = \{4, 6, 8, 9, 10, 12, 14, 15, \ldots\}$
$\overline{X} = \{2, 3, 5, 7, 11, 13, 17, 23, 29, \ldots\}$

NP and co-NP

NP. Decision problems for which there is a poly-time certifier.
Example. SAT, Hamilton cycle, and Composites.

Definition

Given a decision problem X, its complement $\overline{X}$ is the same problem with the yes and no answers reversed.

Example $X = \{4, 6, 8, 9, 10, 12, 14, 15, \ldots\}$
$\overline{X} = \{2, 3, 5, 7, 11, 13, 17, 23, 29, \ldots\}$

co-NP. Complements of decision problems in **NP**.
Example. Un-SAT, No Hamilton cycle, and Primes.

NP = co-NP?

Fundamental open question. Does **NP = co-NP**?

NP = co-NP?

Fundamental open question. Does **NP = co-NP**?

- Do yes instances have succinct certificates iff no instances do?
- Consensus opinion: no.

NP = co-NP?

Shanghai Jiao Tong University

Fundamental open question. Does **NP = co-NP**?

- Do yes instances have succinct certificates iff no instances do?
- Consensus opinion: no.

Theorem

*If **NP** $\neq$ **co-NP**, then **P** $\neq$ **NP**.*

NP = co-NP?

Fundamental open question. Does **NP = co-NP**?

- Do yes instances have succinct certificates iff no instances do?
- Consensus opinion: no.

Theorem

*If **NP** $\neq$ **co-NP**, then **P** $\neq$ **NP**.*

Proof idea.

NP = co-NP?

Fundamental open question. Does **NP = co-NP**?

- Do yes instances have succinct certificates iff no instances do?
- Consensus opinion: no.

Theorem

*If **NP** $\neq$ **co-NP**, then **P** $\neq$ **NP**.*

Proof idea.

- **P** is closed under complementation.
- If **P = NP**, then **NP** is closed under complementation.
- In other words, **NP = co-NP**.
- This is the contrapositive of the theorem.

Good characterizations

Good characterization.[Edmonds 1965] **NP** $\cap$ **co-NP**.

Good characterizations

Good characterization.[Edmonds 1965] **NP** $\cap$ **co-NP**.

- If problem X is in both **NP** and **co-NP**, then:
  - for yes instance, there is a succinct certificate
  - for no instance, there is a succinct disqualifier
- Provides conceptual leverage for reasoning about a problem.

Good characterizations

Good characterization.[Edmonds 1965] **NP** $\cap$ **co-NP**.

- If problem X is in both **NP** and **co-NP**, then:
  - for yes instance, there is a succinct certificate
  - for no instance, there is a succinct disqualifier
- Provides conceptual leverage for reasoning about a problem.

Example. Given a bipartite graph, is there a perfect matching?

Good characterizations

Good characterization.[Edmonds 1965] **NP** $\cap$ **co-NP**.

- If problem X is in both **NP** and **co-NP**, then:
  - for yes instance, there is a succinct certificate
  - for no instance, there is a succinct disqualifier
- Provides conceptual leverage for reasoning about a problem.

Example. Given a bipartite graph, is there a perfect matching?

- If yes, can exhibit a perfect matching.
- If no, can exhibit a set of nodes S such that $|N(S)| < |S|$.

Good characterizations

Observation. $\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{co\text{-}NP}$.

Good characterizations

SHANGHAI JIAO TONG UNIVERSITY

Observation. **P** $\subseteq$ **NP** $\cap$ **co-NP**.

- Proof of max-flow min-cut theorem led to stronger result that max-flow and min-cut are in **P**.
- Sometimes finding a good characterization seems easier than finding an efficient algorithm.

Good characterizations

Observation. **P** $\subseteq$ **NP** $\cap$ **co-NP**.

- Proof of max-flow min-cut theorem led to stronger result that max-flow and min-cut are in **P**.
- Sometimes finding a good characterization seems easier than finding an efficient algorithm.

Fundamental open question. Does **P** $=$ **NP** $\cap$ **co-NP**?

Good characterizations

Shanghai Jiao Tong University

Observation. **P** $\subseteq$ **NP** $\cap$ **co-NP**.

- Proof of max-flow min-cut theorem led to stronger result that max-flow and min-cut are in **P**.
- Sometimes finding a good characterization seems easier than finding an efficient algorithm.

Fundamental open question. Does **P** $=$ **NP** $\cap$ **co-NP**?

- Mixed opinions.
- Many examples where problem found to have a nontrivial good characterization, but only years later discovered to be in **P**.

Factoring is in NP $\cap$ co-NP

Linear programming. Given $A \in \mathcal{R}^{m \times n}, b \in \mathcal{R}^{m}, c \in \mathcal{R}^{n}$, and $\alpha \in R$, does there exist $x \in \mathcal{R}^{n}$ such that $Ax \leq b, x \geq 0$ and $c^T x \geq \alpha$?

Factoring is in NP ∩ co-NP

Linear programming. Given $A \in \mathcal{R}^{m\times n}, b \in \mathcal{R}^m, c \in \mathcal{R}^n$, and $\alpha \in R$, does there exist $x \in \mathcal{R}^n$ such that $Ax \leq b, x \geq 0$ and $c^T x \geq \alpha$?

Theorem (Gale–Kuhn–Tucker 1948)

LINEAR PROGRAMMING ∈ ***NP*** ∩ ***Co-NP*** .

Proof sketch. If (P) and (D) are nonempty, then $\max = \min$.

$$\text{(P)} \max c^T x \quad \text{s.t. } Ax \leq b, \quad x \geq 0$$

$$\text{(D)} \min y^T b \quad \text{s.t. } A^T y \geq c, \quad y \geq 0$$

Factoring is in NP ∩ co-NP

LINEAR PROGRAMMING. Given $A \in \mathcal{R}^{m \times n}, b \in \mathcal{R}^m, c \in \mathcal{R}^n$, and $\alpha \in R$, does there exist $x \in \mathcal{R}^n$ such that $Ax \leq b, x \geq 0$ and $c^T x \geq \alpha$?

Factoring is in NP $\cap$ co-NP

LINEAR PROGRAMMING. Given $A \in \mathcal{R}^{m \times n}, b \in \mathcal{R}^m, c \in \mathcal{R}^n$, and $\alpha \in R$, does there exist $x \in \mathcal{R}^n$ such that $Ax \leq b, x \geq 0$ and $c^T x \geq \alpha$?

Theorem (Khachiyan 1979)

LINEAR PROGRAMMING $\in$ ***P***.

Factoring is in NP $\cap$ co-NP

Theorem (Pratt 1975)

PRIMES $\in$ ***NP*** $\cap$ ***co-NP***.

Factoring is in NP ∩ co-NP

Theorem (Pratt 1975)

PRIMES $\in$ ***NP*** $\cap$ ***co-NP***.

Proof sketch. An odd integer s is prime iff there exists an integer $1 < t < s$ s.t.

$$t^{s-1} \equiv 1 \pmod{s}$$
$$t^{(s-1)/p} \not\equiv 1 \pmod{s}$$

for all prime divisors p of $s - 1$.

Primality testing is in P

Theorem (Agrawal–Kayal–Saxena 2004)

PRIMES $\in$ ***P***.

Factoring is in NP $\cap$ co-NP

Shanghai Jiao Tong University

FACTORIZE. Given an integer x, find its prime factorization.
FACTOR. Given two integers x and y, does x have a nontrivial factor $< y$?

Factoring is in NP $\cap$ co-NP

FACTORIZE. Given an integer x, find its prime factorization.

FACTOR. Given two integers x and y, does x have a nontrivial factor $< y$?

Theorem

FACTOR $\equiv_P$ FACTORIZE

Factoring is in NP $\cap$ co-NP

FACTORIZE. Given an integer x, find its prime factorization.

FACTOR. Given two integers x and y, does x have a nontrivial factor $< y$?

Theorem

FACTOR $\equiv_P$ FACTORIZE

Proof.

Factoring is in NP $\cap$ co-NP

FACTORIZE. Given an integer x, find its prime factorization.
FACTOR. Given two integers x and y, does x have a nontrivial factor $< y$?

Theorem

FACTOR $\equiv_P$ FACTORIZE

Proof.

- $\leq_P$ trivial.
- $\geq_P$ binary search to find a factor; divide out the factor and repeat.

Factoring is in NP $\cap$ co-NP

SHANGHAI JIAO TONG UNIVERSITY

FACTORIZE. Given an integer x, find its prime factorization.
FACTOR. Given two integers x and y, does x have a nontrivial factor $< y$?

Theorem

FACTOR $\equiv_P$ FACTORIZE

Proof.

- $\leq_P$ trivial.
- $\geq_P$ binary search to find a factor; divide out the factor and repeat.

Theorem

FACTOR $\in$ ***NP*** $\cap$ ***co-NP***.

Factoring is in NP ∩ co-NP

SHANGHAI JIAO TONG UNIVERSITY

FACTORIZE. Given an integer x, find its prime factorization.
FACTOR. Given two integers x and y, does x have a nontrivial factor $< y$?

Theorem

FACTOR $\equiv_P$ FACTORIZE

Proof.

- $\leq_P$ trivial.
- $\geq_P$ binary search to find a factor; divide out the factor and repeat.

Theorem

FACTOR $\in$ ***NP*** $\cap$ ***co-NP***.

Proof.

Factoring is in NP ∩ co-NP

FACTORIZE. Given an integer x, find its prime factorization.
FACTOR. Given two integers x and y, does x have a nontrivial factor $< y$?

Theorem

FACTOR $\equiv_P$ FACTORIZE

Proof.

- $\leq_P$ trivial.
- $\geq_P$ binary search to find a factor; divide out the factor and repeat.

Theorem

FACTOR $\in$ ***NP*** $\cap$ ***co-NP***.

Proof.

- Certificate: a factor p of x that is less than y.

Factoring is in NP ∩ co-NP

SHANGHAI JIAO TONG UNIVERSITY

FACTORIZE. Given an integer x, find its prime factorization.

FACTOR. Given two integers x and y, does x have a nontrivial factor $< y$?

Theorem

FACTOR $\equiv_P$ FACTORIZE

Proof.

- $\leq_P$ trivial.
- $\geq_P$ binary search to find a factor; divide out the factor and repeat.

Theorem

FACTOR $\in$ ***NP*** $\cap$ ***co-NP***.

Proof.

- Certificate: a factor p of x that is less than y.
- Disqualifier: the prime factorization of x (where each prime factor is greater than y).

Is factoring in P?

Shanghai Jiao Tong University

Fundamental question. Is FACTOR $\in$ **P**?

Is factoring in P?

Fundamental question. Is FACTOR ∈ **P**?

Challenge. Factor this number.

74037563479561712828046796097429573142593188889231289
08493623263897276503402826627689199641962511784399589
43305021275853701189680982867331732731089309005525051
16877063299072396380786710086096962537934650563796359

RSA-704

($30,000 prize if you can factor)

Exploiting intractability

Modern cryptography.

- Example. Send your credit card to Amazon.
- Example. Digitally sign an e-document.
- Enables freedom of privacy, speech, press, political association.

Exploiting intractability

Shanghai Jiao Tong University

Modern cryptography.

- Example. Send your credit card to Amazon.
- Example. Digitally sign an e-document.
- Enables freedom of privacy, speech, press, political association.

RSA. Based on dichotomy between complexity of two problems.

- To use: generate two random n-bit primes and multiply.
- To break: suffices to factor a $2n$-bit integer.

Factoring on a quantum computer

Shanghai Jiao Tong University

Theorem (Shor 1994)

Can factor an n-bit integer in $O(n^3)$ steps on a "quantum computer".

Factoring on a quantum computer

Theorem (Shor 1994)

Can factor an n-bit integer in $O(n^3)$ steps on a "quantum computer".

2001. Factored $15 = 3 \times 5$ (with high probability) on a quantum computer.
2012. Factored $21 = 3 \times 7$.

Factoring on a quantum computer

Theorem (Shor 1994)

Can factor an n-bit integer in $O(n^3)$ steps on a "quantum computer".

2001. Factored $15 = 3 \times 5$ (with high probability) on a quantum computer.
2012. Factored $21 = 3 \times 7$.

Fundamental question. Does **P = BQP**?

NP-hard

A note on terminology: consensus

NP-complete. A problem in **NP** such that every problem in **NP** poly-time reduces to it.

A note on terminology: consensus

NP-complete. A problem in **NP** such that every problem in **NP** poly-time reduces to it.

NP-hard. [[Bell Labs, Steve Cook, Ron Rivest, Sartaj Sahni] A problem such that every problem in **NP** poly-time reduces to it.