Algorithms for Data Processing
Lecture VIII: Intractable Problems–NP Problems

Alessandro Artale

Free University of Bozen-Bolzano
Faculty of Computer Science
http://www.inf.unibz.it/˜artale
artale@inf.unibz.it

2019/20 – First Semester
MSc in Computational Data Science — UNIBZ

Some material (text, figures) displayed in these slides is courtesy of:
Alberto Montresor, Werner Nutt, Kevin Wayne, Jon Kleinberg, Eva Tardos.

Problems and Algorithms – Decision Problems

The Complexity Theory considers so called Decision Problems.

Decision Problem.

- Input encoded as a finite binary string s;
- Decision Problem X: Is conceived as a set of strings on which the answer to the decision problem X is *"yes"*;
- Algorithm A for a decision problem X receives an input string s, and

$$A(s) = \begin{cases} yes & if\ s \in X \\ no & if\ s \notin X \end{cases}$$

Problems and Algorithms – Decision Problems

The Complexity Theory considers so called Decision Problems.

Decision Problem.

- Input encoded as a finite binary string s;
- Decision Problem X: Is conceived as a set of strings on which the answer to the decision problem X is *"yes"*;
- Algorithm A for a decision problem X receives an input string s, and

$$A(s) = \begin{cases} yes & \text{if } s \in X \\ no & \text{if } s \notin X \end{cases}$$

Definition. **P** = set of decision problems for which there exists a poly-time algorithm.

Towards NP — Efficient Verification

- The issue here is the contrast between finding a solution Vs. checking a proposed solution.
- Consider for example `3-SAT`:
  - We do not know a polynomial-time algorithm to find solutions; but
  - Checking a proposed solution can be easily done in polynomial time (just plug 0/1 and check if it is a solution).

Towards NP — Efficient Verification/2

Formalize the idea that a solution to a problem can be checked efficiently.

- Checking Algorithm for a problem X: Checks whether t is a solution for a given input s of problem X;
- t is called the certificate or witness that contains the evidence that s is a *"yes"* instance of X.

Towards NP — Efficient Verification/2

Formalize the idea that a solution to a problem can be checked efficiently.

- Checking Algorithm for a problem X: Checks whether t is a solution for a given input s of problem X;
- t is called the certificate or witness that contains the evidence that s is a *"yes"* instance of X.

Definition. An algorithm $C(s,t)$ is an efficient certifier for a problem X if:

- $C(s,t)$ runs in polynomial time, and
- For every string s, we have $s \in X$ if there exists a string t (the certificate) such that $|t| \leq p(|s|)$ ($p()$ polynomial function), and $C(s,t) = yes$.

Towards NP — Efficient Verification/3

Definition. An algorithm $C(s, t)$ is an efficient certifier for a problem X if:

- $C(s, t)$ runs in polynomial time, and
- For every string s, we have $s \in X$ if there exists a string t (the certificate) such that $|t| \leq p(|s|)$ ($p()$ polynomial function), and $C(s, t) = yes$.

An efficient certifier $C(s, t)$:

- It is not deciding whether an input s belongs to X, but
- It is efficiently evaluating whether a given t is a *certificate* for s to belong to X.
- It can be used as an exponential *brute force* algorithm.

The NP Class of Problems

Definition. NP = set of decision problems for which there exists an efficient certifier.

Note: NP stands for Nondeterministic Polynomial time.

The NP Class of Problems

Definition. NP = set of decision problems for which there exists an efficient certifier.

Note: NP stands for Nondeterministic Polynomial time.

We can observe immediately that:

Theorem. $P \subseteq NP$.
Proof. Let A be a polynomial time algorithm that solves X. Then, choose $t = \epsilon$ and $C(s,t) \equiv A(s)$.

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 $x_1 = \mathit{true},\ x_2 = \mathit{true},\ x_3 = \mathit{false},\ x_4 = \mathit{false}$

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

$\mathsf{P} \subseteq \mathsf{NP} \subseteq \textsc{ExpTime}$

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

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

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

Theorem. $\mathsf{P} \subseteq \mathsf{NP}$.

P $\subseteq$ NP $\subseteq$ ExpTime

P. Decision problems for which there exists a poly-time algorithm.
NP. Decision problems for which there exists a poly-time certifier.
ExpTime. Decision problems for which there exists an exponential-time algorithm.

Theorem. P $\subseteq$ NP.

Theorem. NP $\subseteq$ ExpTime.
Proof. Consider any problem $X \in$ NP.

- By definition, there exists a poly-time certifier $C(s, t)$ for X, where $|t| \leq p(|s|)$ for some polynomial $p()$;
- To solve instance s, run $C(s, t)$ on all strings t with $|t| \leq p(|s|)$ (exponentially many).
- Return *yes* iff $C(s, t)$ returns *yes* for at least one of these potential certificates.

P Vs. NP

Facts.

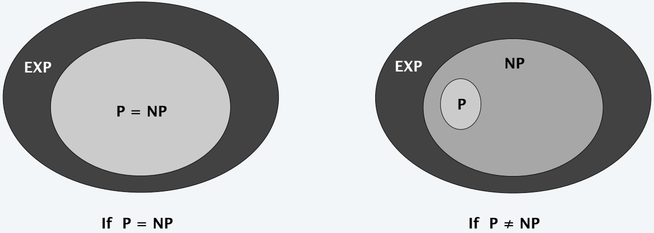
1. $\mathrm{P} \subseteq \mathrm{NP} \subseteq \textsc{ExpTime}$;
2. $\mathrm{P} \neq \textsc{ExpTime}$, then:

either $\mathrm{P} \neq \mathrm{NP}$, or $\mathrm{NP} \neq \textsc{ExpTime}$, or both.

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 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.

Possible outcomes

P ≠ NP

" I conjecture that there is no good algorithm for the traveling salesman problem. My reasons are the same as for any mathematical conjecture: (i) It is a legitimate mathematical possibility and (ii) I do not know."

— *Jack Edmonds 1966*

" In my view, there is no way to even make intelligent guesses about the answer to any of these questions. If I had to bet now, I would bet that P is not equal to NP. I estimate the half-life of this problem at 25–50 more years, but I wouldn't bet on it being solved before 2100. "

— *Bob Tarjan (2002)*

Possible outcomes

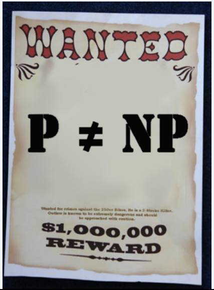
P = NP

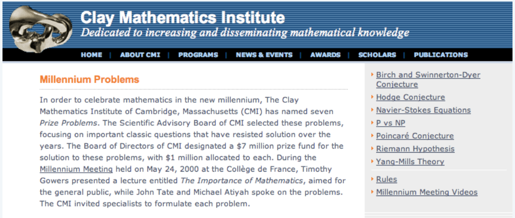
> “ *I think that in this respect I am on the loony fringe of the mathematical community: I think (not too strongly!) that P=NP and this will be proved within twenty years. Some years ago, Charles Read and I worked on it quite bit, and we even had a celebratory dinner in a good restaurant before we found an absolutely fatal mistake.* ”
>
> — *Béla Bollobás (2002)*

> “ *In my opinion this shouldn’t really be a hard problem; it’s just that we came late to this theory, and haven’t yet developed any techniques for proving computations to be hard. Eventually, it will just be a footnote in the books.* ” — *John Conway*

Millennium prize

Millennium prize. $1 million for resolution of **P ≠ NP** problem.

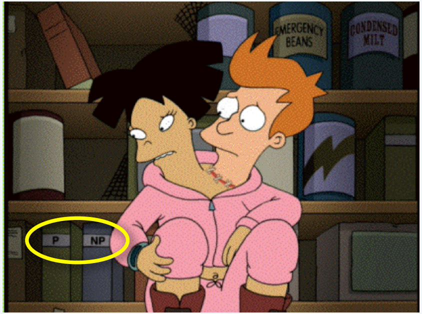




NP-completeness and pop culture

Some writers for the Simpsons and Futurama.

- J. Steward Burns. *M.S. in mathematics (Berkeley '93)*.
- David X. Cohen. *M.S. in computer science (Berkeley '92)*.
- Al Jean. *B.S. in mathematics. (Harvard '81)*.
- Ken Keeler. *Ph.D. in applied mathematics (Harvard '90)*.
- Jeff Westbrook. *Ph.D. in computer science (Princeton '89)*.



NP-complete Problems

Fundamental Question: What are the hardest problems in NP?

NP-complete Problems

Fundamental Question: What are the hardest problems in NP?

Definition. A problem X is said **NP-complete** if:

1. $X \in \text{NP}$, and
2. For any $Y \in \text{NP}$, $Y \leq_P X$.

Thus, X is as hard as any other NP problem!

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.

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

by definition of **NP**-complete

by assumption

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

NP-complete Problems/2

Theorem. Let $X \in$ NP-complete problems. Then, X is solvable in polynomial time if and only if P $=$ NP.

($\Leftarrow$) If P $=$ NP, then $X \in$ P because $X \in$ NP.

($\Rightarrow$) Let X be solvable in polynomial time. Since X is NP-complete, then

- For any $Y \in$ NP, $Y \leq_P X$, and thus Y is solvable in polynomial time, thus
- NP $\subseteq$ P, and since we already proved that P $\subseteq$ NP, we finally obtain
- P $=$ NP.

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 $n/\log n$ distinct function and predicate symbols. The logical connectives are & (and), $\vee$ (or), and $\neg$(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 $u \in T$ and the no state if $u \notin T$. 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 $w \in S$.

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.

Circuit Satisfiability Problem

Definition. A Circuit C is a labeled, directed acyclic graph where:

- Nodes with no incoming edges (later called inputs) are labeled either with one of the constants 0 or 1, or with the name of a distinct variable;
- Internal nodes are labeled with one of the Boolean operators $\wedge, \vee, \neg$;
- There is a single node with no outgoing edges, representing the output (the result that is computed by the circuit.)

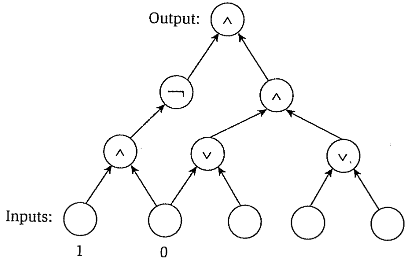


Figure 8.4 A circuit with three inputs, two additional sources that have assigned truth values, and one output.

Circuit Satisfiability: The First NP-Complete Problem

Theorem. [Cook 1971, Levin 1973] Circuit Satisfiability is NP-complete.
[Proof Sketch]

- Show that given an arbitrary problem $\texttt{X} \in \text{NP}$, then $\texttt{X} \leq_{\text{P}}$ `Circuit Satisfiability`.
- Main Idea: Show that any algorithm that takes a fixed number n of bits as input and produces a yes/no answer can be represented by a circuit.
- This circuit should output 1 on precisely the inputs for which the algorithm outputs yes.
- If the algorithm takes $p(n)$ steps, then the circuit has polynomial size.
- We are not showing this construction but we see how it can be used in the proof.

Circuit Satisfiability: The First NP-Complete Problem/2

Theorem. [Cook 1971, Levin 1973] Circuit Satisfiability is NP-complete.
[Proof Sketch/2]

- Since $\mathtt{X} \in$ NP it has an efficient certifier, thus
- To determine whether $s \in \mathtt{X}$, for some input s of size n, we need to answer the following question:

  Is there a certificate t of length $p(n)$ so that $C(s, t) =$ yes?
- We view $C(s, t)$ as an algorithm on $n + p(n)$ bits: the input s and the certificate t, and
- Convert it to a polynomial-size circuit C with $n + p(n)$ sources.
- The first n sources will be hard-coded with the values of the bits in s, and
- The $p(n)$ sources will be labeled with variables representing the bits of t; these latter sources will be the inputs to C.
- $s \in \mathtt{X}$ if and only if the circuit C is satisfiable.

More hard computational problems

Garey and Johnson. Computers and Intractability.

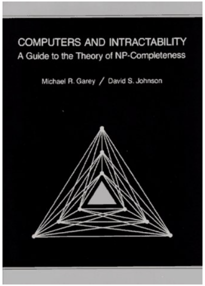
- Appendix includes over 300 **NP**-complete problems.
- Most cited reference in computer science literature.

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

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, ..., 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.

You NP-complete me

Graph k-Coloring

While 2-Coloring (Bipartite Graphs) is a P-Time problem, checking whether a graph is 3-colorable is a hard problem.

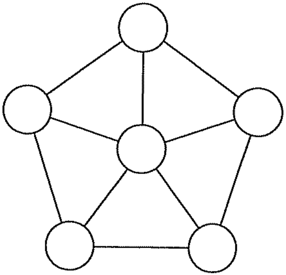
Graph k-colorability. Given a graph G and an integer k assign a color to each node of G so that if (u, v) is an edge, then u and v are assigned different colors from the k available colors.

Applications. Graph colorability is a problem that arises naturally whenever one is trying to allocate resources in the presence of conflicts.

- e.g., assign one of k transmitting wavelengths to each of n devices; but if two devices are sufficiently close to each other, then they need to be assigned different wavelengths to prevent interference.

Graph k-Coloring/2

- Fact 1. There is not fixed constant k so that every graph is k-colorable.
  - For example, take a set of n nodes and join each pair of them by an edge, the resulting graph needs n colors.
- Fact 2. No simple efficient algorithm for the 3-Coloring Problem exists.
  - The following graph is not 3-colorable but does not have a cicle of 4 nodes mutually connected.



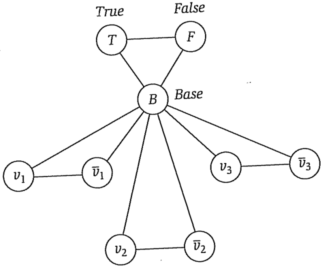
3-Coloring is NP-complete

Theorem. Graph 3-coloring in an NP-complete problem.

Graph 3-coloring is in NP. Certificate: a k-coloring of the graph. We can check in polynomial time whether $k \leq 3$ and that every edge in the graph has endpoints with different colors.

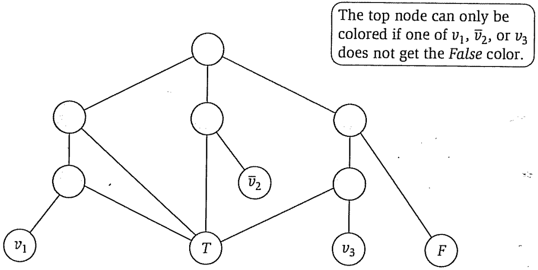
Show that 3-SAT $\leq_P$ 3-COLORING.

Instance Construction. We first construct the following graph G (here is the case with 3 variables):



3-Coloring is NP-complete/2

- Extend G so that there is a satisfying assignments if and only if the full graph is 3-colorable.
- To each set of 3 nodes in a Clause we a attach a 6-node graph. For example, consider the clause $C = x_1 \vee \overline{x_2} \vee x_3$



3-Coloring is NP-complete/3

- ($\Leftarrow$, Soundness) Suppose G is 3-colorable.
- Each node v_i is assigned either the *True* or the *False* color (see first graph);
- We set the assignment to x_i accordingly.
- As we said above, there must be in every clause at least one variable set to *True* for otherwise G is not 3-colorable (a contradiction).
- ($\Rightarrow$, Completeness) Let 3-SAT be satisfiable, then the resulting graph is 3-colorable.
- Can be showed by a case analysis and by setting to the True color the variable assigned to 1.

P or NP-complete

There are Problems in NP which are not known to be in **P nor in the class of NP-complete problems.**

Theorem [Ladner 1975] Unless P = NP, there exist problems in NP that are in neither P nor NP-complete.

NP-intermediate. `GRAPH-ISOMORPHISM`, `INTEGER-FACTORIZATION`, **etc.**

Asymmetry of NP

Observation. The definition of efficient certification, and hence of NP, is asymmetric.

- YES instances. An input string s is a yes instance if and only if there exists a polynomially bounded t so that $C(s, t) = yes$.

By negating the above statement we get:

- NO instances. An input string s is a no instance if and only if for all polynomially bounded t, it is the case that $C(s, t) = no$.
  - For a no instance, no short proof is guaranteed by the definition.

Asymmetry of NP: SAT vs. UN-SAT

UN-SAT. Given a CNF formula Φ, is there NO satisfying truth assignment?

SAT vs. UN-SAT

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

Complement Problems

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

The following are examples of complementary problems:

- SAT Vs. UN-SAT
- VERTEX-COVER Vs. NO-VERTEX-COVER
- etc.

co-NP. Complements of decision problems in NP.

P Vs. co-P

Theorem. The class **P is closed under complementation, i.e., P= co-P.**

Proof. Let $X \in \mathsf{P}$ and A_X be a polynomial algorithm solving the decision problem X. Then, the algorithm $\overline{A_X}$ that runs A and inverts the *yes/no* answers, is a polynomial algorithm for $\overline{X}$.

NP Vs. co-NP

Observation. When $X \in$ NP it is not so clear to see whether $\overline{X} \in$ NP.

- $\overline{X}$ has a different nature: An input string $s \in \overline{X}$ if and only if for all polynomially bounded t, it is the case that $C(s, t) = no$.
  - It is not enough to invert the answer of the efficient certifier C to get a certifier $\overline{C}$ for $\overline{X}$.
  - The critical point is the shift from *there exists* t to *for all* t.

NP Vs. co-NP

Observation. When $X \in \text{NP}$ it is not so clear to see whether $\overline{X} \in \text{NP}$.

- $\overline{X}$ has a different nature: An input string $s \in \overline{X}$ if and only if for all polynomially bounded t, it is the case that $C(s, t) = no$.
  - It is not enough to invert the answer of the efficient certifier C to get a certifier $\overline{C}$ for $\overline{X}$.
  - The critical point is the shift from *there exists* t to *for all* t.

Open Question. Does NP = co-NP?

- Consensus opinion: no.

NP Vs. co-NP/2

Open Question. Does NP = co-NP?

- Consensus opinion: no.

Theorem. If NP $\neq$ co-NP, then **P $\neq$ NP.**
Proof. We show the contrapositive, i.e., if **P = NP, then, NP = co-NP.**

- Since **P is closed under complementation, then, if** P = NP**, then,** NP **would be closed under complementation as well.**

Good Characterizations: The Class NP $\cap$ co-NP

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

- If problem X is in both NP and co-NP, then:
  - for a *yes* instance, there is a succinct certificate;
  - for a *no* instance, there is a succinct disqualifier.
- Problems for wich there is always a nice certificate for the solution.

Good Characterizations: The Class NP $\cap$ co-NP/2

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

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**.

Primality testing is in NP ∩ co-NP

Theorem. [Pratt 1975] PRIMES $\in$ **NP** $\cap$ **co-NP**.

SIAM J. COMPUT.
Vol. 4, No. 3, September 1975

EVERY PRIME HAS A SUCCINCT CERTIFICATE*

VAUGHAN R. PRATT†

Abstract. To prove that a number n is composite, it suffices to exhibit the working for the multiplication of a pair of factors. This working, represented as a string, is of length bounded by a polynomial in $\log_2 n$. We show that the same property holds for the primes. It is noteworthy that almost no other set is known to have the property that short proofs for membership or nonmembership exist for all candidates without being known to have the property that such proofs are easy to come by. It remains an open problem whether a prime n can be recognized in only $\log_2^\alpha n$ operations of a Turing machine for any fixed α.

The proof system used for certifying primes is as follows.

AXIOM. $(x, y, 1)$.

INFERENCE RULES.

$$R_1: \quad (p, x, a), q \vdash (p, x, qa) \quad \text{provided } x^{(p-1)/q} \not\equiv 1 \pmod p \text{ and } q \mid (p-1).$$

$$R_2: \quad (p, x, p-1) \vdash p \quad \text{provided } x^{p-1} \equiv 1 \pmod p.$$

THEOREM 1. *p is a theorem $\equiv$ p is a prime.*

THEOREM 2. *p is a theorem $\supset$ p has a proof of $\lceil 4 \log_2 p \rceil$ lines.*

Primality testing is in P

Theorem. [Agrawal–Kayal–Saxena 2004] PRIMES ∈ **P**.

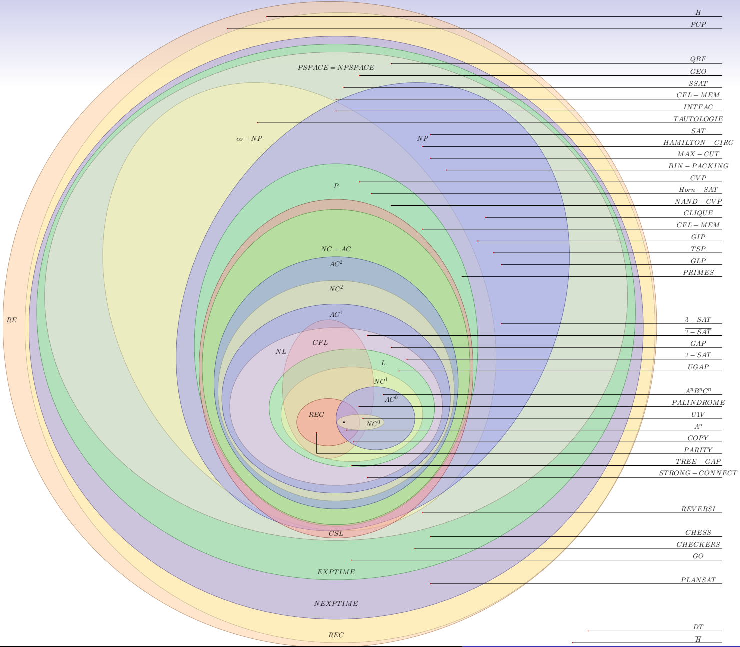
Annals of Mathematics, **160** (2004), 781–793

PRIMES is in P

By Manindra Agrawal, Neeraj Kayal, and Nitin Saxena*

Abstract

We present an unconditional deterministic polynomial-time algorithm that determines whether an input number is prime or composite.



COMPLEXITY DIAGRAM

Definitions

The Comlexity Classes

- $NONE = \emptyset$
- $NC^0 =$ Functions computable by constant-depth, bounded-fanin circuits.
- $AC^0 =$ Class of decision problems solvable by a nonuniform family of Boolean circuits, with polynomial size, depth $O(log^i(n))$, and unbounded fanin.
- $REG = DSPACE(O(1))$: The decision problems that can be solved in constant space
- $NC^1 =$ Class of decision problems solvable by a nonuniform family of Boolean circuits, with polynomial size, depth $O(log(n))$, and fan-in 2.
- $CFL =$ Context-Free Languages
- $AC^1 =$ Class of decision problems solvable by a nonuniform family of Boolean circuits, with polynomial size, depth $O(log(n))$
- $L =$ Class of decision problems solvable by a Turing machine restricted to use an amount of memory logarithmic in the size of the input.
- $NL = coNL$ Class of decision problems solvable by a nondeterministic Turing machine restricted to use an amount of memory logarithmic in the size of the input.
- $NC = AC$: Decidable in polylogarithmic time on a parallel computer with a polynomial number of processors.
- $CSL = NLINSPACE$ The class of languages generated by context-sensitive grammars.
- $P = Polysize(uniform) =$ The class of decision problems solvable in polynomial time by a Turing machine.
- NP = Nondeterministic Polynomial-Time
- $coNP$= Complement of NP
- $PSPACE = NPSPACE = AP = APP = IP$
- $EXPTIME = \bigcup_{k\in\mathbb{N}} DTIME(2^{n^k})$
- $NEXPTIME = \bigcup_{k\in\mathbb{N}} NTIME(2^{n^k})$
- REC= Class of decidable Problems
- RE is the class of decision problems for which a „yes" answer can be verified by a Turing machine in a finite amount of time. (If the answer is „no," on the other hand, the machine might never halt.)

Some Missing Classes

- $EEXPTIME$
- $NEEXPTIME$
- $EXPSPACE$
- BPP
- BQP

The Languages

- $A^n = \{x|x \in \{a\}^*\} \in REG$
- $A^nB^n = \{a^nb^n|n \geq 1\} \in CFL$
- $A^nB^nC^n = \{a^nb^nc^n|n \geq 1\} \in CSL$
- $BIN - PACKING$ =Objects of different volumes must be packed into a finite number of bins of capacity V in a way that minimizes the number of bins used. $\in NP - c$
- $DT := \{i|\Phi_i^1\text{is total}\} \notin RE$
- $CHECKERS =$ The generalized version of the game $\in EXPTIME - c$
- $CHESS =$ The generalized version of the game chess $\in EXPTIME - c$
- $CFL - MEM =$ Context Free Grammar Membership
- $CLIQUE =$ If a graph contains a clique of at least a given size k. $\in NP - c$
- $COPY = \{uv|u = v\}$
- $CVP =$ Circuit Value Problem $\in P - c$
  - $NAND - CVP = CVP$ only using NAND-components $\in P$
- $GAP = PATH = \{\langle D, s, t\rangle|$Dir. Graph D has path from $s$to $t\} \in NL - c$
  - $UGAP = GAP$ in an underícted graph $\in L - c$
  - $TREE - GAP = GAP$ in a tree (DAG) $\in L - c$
- $GEO \in PSCACE - c$
- $GIP =$ Graph isomorphism problem: Determining whether two graphs are isomorphic $\in NP$
- $GLP = \{(A, b) \in \mathbb{Z}^{m,n} \times \mathbb{Z}^m|\exists x \in \mathbb{Z}^n : Ax \leq b\} \in NP - c$
- $GO =$ The generalized version of the japanese game Go $\in EXPTIME - c$
- $H := \{\langle i, x\rangle|x \in dom(\Phi_i^1)\} \in RE - c$
  - $\overline{H} = \{\langle i, x\rangle|x \notin H\} \notin RE$
- $HAMILTON - CIRC =$ If A graph contains a hamilton circle. $\in NP - c$
- $INTFAC =$ integer factorization is the process of breaking down a composite number into smaller non-trivial divisors, which when multiplied together equal the original integer. $\in NP \cap co - NP$
- $MAX - CUT =$ The Maximum Cut of a Graph. $\in NP - c$
- $PALINDROME =$ if a word is a palindrome. $\in CFL$
- $PARITY = \{a \in \{0, 1\}^*|\#1 \text{ is odd}\} \in REG$
- $PCP =$ Post correspondence problem $\in NP - c$
- $PLANSAT =$ set off all strings $s \subseteq L^*$ such that s is the statement of a solvable problem. $\in NEXPTIME - c$ (Without certain restrictions the problem is $\in EXPSPACE - c$.)
- $PRIMES =$ Determining whether an input number is prime. $\in P$
- $QBF =$ Satisfiability problem for quantified boolean formula $\in PSPACE - c$
- $REVERSI =$ Generalized Version of the board game. $\in PSPACE - c$
- $SAT =$ Boolean satisfiability problem $\in NP - c$
  - $2 - SAT =$ SAT with CNF with 2 variables per clause $\in NL - c$
  - $\overline{2 - SAT} = \{x|x \notin 2 - SAT\} \in NL - c$
  - $3 - SAT =$ SAT with CNF with 3 variables per clause $\in NP - c$
  - $Horn - SAT =$ SAT with a conjunction of Horn clauses. $\in P - c$
  - $SSAT =$ Probabilistic satisfiability problem $\in PSPACE - c$
- $STRONG - CONNECT \in NL$
- $TAUTOLOGY =$ If a given boolean formula is a tautology. $\in co - NPc$
- $TSP =$ Traveling sales person $\in NP - c$
- $U1V = \{u1v||u| = |v|\}$

Relations

Unsolved Problems

- $NL \subsetneq EXPTIME$
  This includes the questions of „$NL \neq P$?", „$P \neq NP$?", „$PSAPCE \neq P$?" and „$EXPTIME \neq PSAPCE$?"
- Is NC propper? $\Leftrightarrow NC^i \neq NC^{i+1}$? (and is AC propper?)
- Is EXPTIME=NEXPTIME? (A direct result of P=NP would be EXPTIME=NEXPTIME)

Known Subsets

- Chomsky hierarchy:
  $REG = TYPE3 \subset DCFL \subset CFL = TYPE2 \subset CSL = TYPE1 \subset RE = TYPE0$
- $NC^0 \subset AC^0 \subset NC^1 \subset L \subset NL \subsetneq NC = AC \subsetneq P \subsetneq NP \subsetneq PSPACE \subsetneq EXPTIME \subsetneq NEXPTIME \subsetneq EXPSPACE = NEXPSAPCE \subsetneq REC \subsetneq RE$
- $NL \subsetneq PSPACE \subsetneq EXPSPACE$
- $NC \subsetneq PSPACE$ and $NC \subsetneq CSL$

Savich's Theorem

For any function $f(n) \geq log(n)$: $DSPACE(f(n)) \subsetneq NSPACE(f(n))$
$\Rightarrow L \neq NL$, $PSPACE = NPSPACE$, $EXPSPACE = NEXPSPACE$, ...

Thank You!