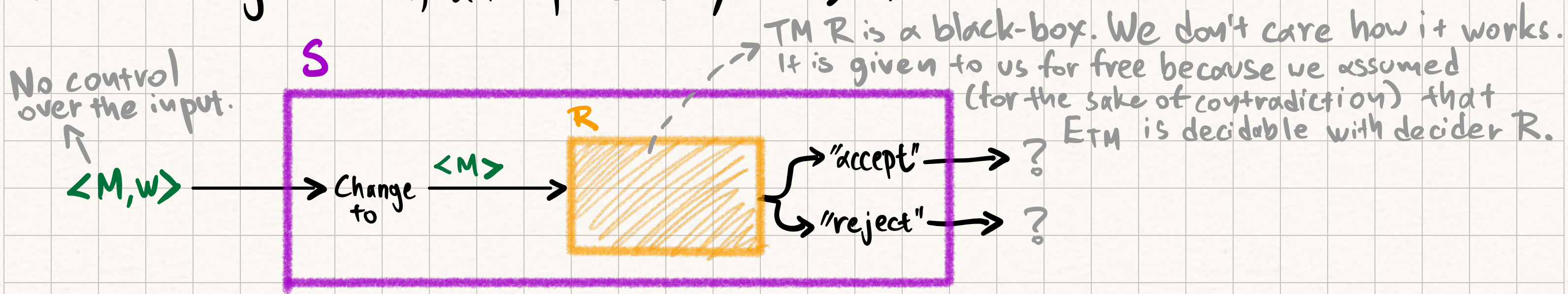
Theorem 5.2. Language $E_{TM} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$ is undecidable

Proof (Step-by-step):

- As always, we proceed with a proof by contradiction. Suppose for the sake of contradiction that E_{TM} is decidable, then there exists a decider R for E_{TM}.

- As a next step, we will choose a language that we know is undecidable, e.g. L_{TM}, and construct a new TM S that ① uses R internally
② is a decider for (the undecidable) language L_{TM}.

Attempt-1: Let's first consider the simplest possible case. Since S needs to process inputs for L_{TM}, it expects inputs of the form $\langle M, w\rangle$. Notice though that R, being a decider for E_{TM}, expects inputs of the form $\langle M\rangle$. Then, the about to be constructed S can receive $\langle M, w\rangle$, then eliminate the second argument w, and pass only $\langle M\rangle$ to R.



Case Analysis: If R returns "accept", then we know that the given input $\langle M\rangle$ does not accept any string, i.e., $\langle M\rangle \in E_{TM}$. Since no input can make M accept, we know for a fact that M does not accept (it may reject or loop) the originally passed input w. Therefore, in that case S can safely "accept".
But we have not completed the proof that the proposed S is a decider. We have only covered the R-accept case.

If R returns "reject", then we know that the given input $\langle M\rangle$ accepts at least one input string, i.e., $\langle M\rangle \notin E_{TM}$. But we have no idea which string(s) is accepted by M. In fact we don't even know if $M(w)$ even halts! So at this point we are stuck. The proposed S construction is not giving any useful insight on whether $M(w)$ accepts or not.

Attempt-2: We have to come up with a non-obvious construction for S. The goal is still for S to: ① use R internally
② be a decider for (the undecidable) language L_{TM}.

But we have to take a less direct route.

- Instead, we will come up with a brand new M_1 that can be fed to the decider R and can help us understand if the input machine M accepts the input string x.
If we have such an M_1, then S can be a TM that takes as an input $\langle M, x\rangle$ constructs M_1 on-the-fly, feeds M_1 to R, and outputs a decision.
M_1 is a function of input M
Given that M is not known without seeing the input, M_1 has to be constructed on-the-fly.

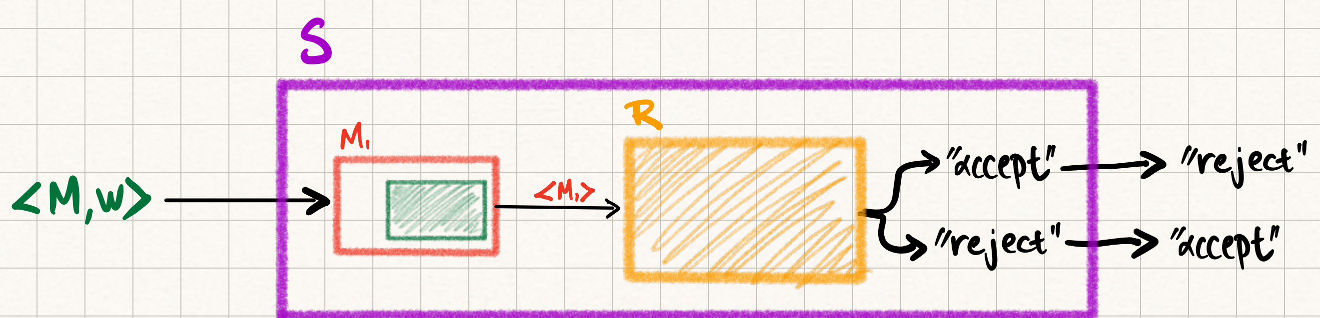
How do we come up with M_1? Recall that in Attempt-1 the problem was that if R outputs "reject" then we don't know if M accepts on w or a different input string. Let's try to fix this!

- We want to define M_1 in such a way that when we feed it to decider R, i.e., $R(\langle M_1 \rangle)$, we will now that if we see "reject" then M accepts on w.

- This means that M_1 should not accept any string that is not w. In case, that the input string is in fact w, it should output whatever M would output on w.

S = "On input $\langle M, w\rangle$, an encoding of a TM M and an input string w:
1. Construct the following machine M_1:
   M_1 = "On input x:
   1. If $x \neq w$, then M_1 rejects
   2. If $x = w$, then run M on w and accept if $M(w)$ accepts."
2. Run R on input $\langle M_1 \rangle$.
3. If R accepts, then S rejects.
4. If R rejects, then S accepts.

For the last part of the proof you have to show that the proposed TM S is indeed a decider for L_{TM}. (Finish the proof at home)



Complexity Theory

Even if we show that a problem is decidable (in theory) it doesn't necessarily mean that it is efficient to calculate its solution in practice.
Thus, we study computational complexity theory which analyzes the amount of resources required for solving problems.

Quick Refresher

- In worst-case analysis, we consider the longest running time of all inputs of a fixed length.
- In average-case analysis, we consider the average of all the running times of inputs of a particular length.

Definition 7.1: Let M be a deterministic TM that halts on all inputs. The running time or time complexity of M is the function $f: \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that M uses on any input of length n.

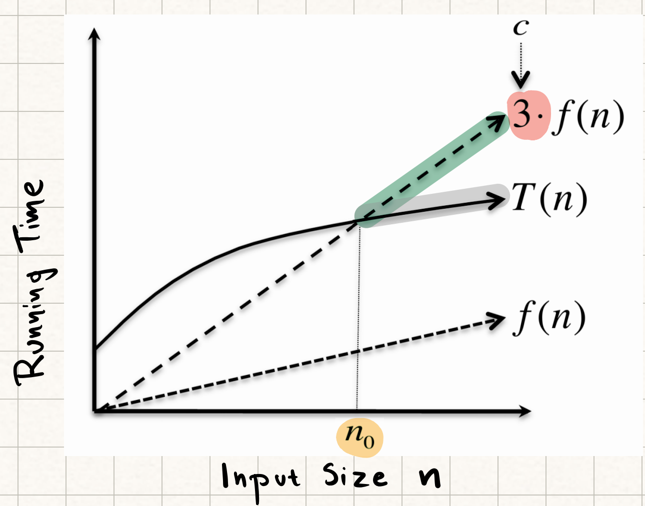
- If $f(n)$ is the running time of M, we say that M runs in time $f(n)$.

- To avoid complex expressions of the running time, we prefer to estimate it using asymptotic analysis. Focus on the highest-order term of the running time expression.

Definition 7.2: Let f and g be functions $f, g: \mathbb{N} - \mathbb{R}^{+}$. We say that $f(n) = O(g(n))$ if positive integers c and n_0 exist such that for every $n \geq n_0$:

$$f(n) \leq c \cdot g(n).$$

When $f(n) = O(g(n))$, we say $g(n)$ is an upper-bound for $f(n)$, i.e., $g(n)$ is an asymptotic upper bound for $f(n)$.



Time Complexity (Sipser, Chapter 7.1)

Take the language $A = \{0^k 1^k \mid k \geq 0\}$. We know that A is decidable but how much time does a single-tape Turing machine need to decide A?

Let's see a low-level description of a TM M_1 that decides A.

left-to-right scan takes n steps

M_1 = "On input string w:
1. Scan across the tape and "reject" if a 0 is found to the right of a 1.
2. Repeat if both 0s and 1s remain on the tape → $\frac{n}{2}$ repetitions
3. Scan across the tape, crossing off a single 0 and a single 1. Each scan takes n steps
4. If 0s still remain after all the 1s have been crossed off, or if 1s still remain after all the 0s have been crossed off, then "reject". Otherwise, "accept"

Rewind the tape to the start takes n steps

In total, M_1 is a $O(n^2)$ Turing machine.

Definition 7.7: Let $t: \mathbb{N} \to \mathbb{R}^+$ be a function. Define the **time complexity class** **TIME(t(n))**, to be the collection of all languages that are decidable by an $O(t(n))$ time Turing machine.

Thus, the analysis for M_1 shows that $A \in TIME(n^2)$, because $TIME(n^2)$ contains all languages that can be decided in $O(n^2)$ time.

M_2 = "On input string w:
1. Scan across the tape and "reject" if a 0 is found to the right of a 1.
2. Repeat if both 0s and 1s remain on the tape
3. Scan across the tape, checking whether the total number of 0s and 1s remaining is even or odd. If it is odd, "reject"
4. Scan again across the tape, crossing off every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1.
5. If no 0s and no 1s remain on the tape, "accept". Otherwise, "reject"

Quiz 5.1: Based on the time complexity of M_2, we can say that (pick the most accurate upper bound):

A) $A \in TIME(n^2)$
B) $A \in TIME(n \log n)$
C) $A \in TIME(n)$
D) $A \in TIME(\log n)$

Suppose we have access to a TM with two tapes. We can design a two-tape approach with TM M_3

M_3 = "On input string w:
1. Scan across tape A and reject is found to the right of a 1
2. Scan across the 0s on tape A until the first 1. At the same time, copy the 0s onto tape B.
3. Scan across the 1s on tape A until the end of the input. For each 1 read on tape A, cross off a 0 on tape B. If all 0s are crossed off before all the 1s are read, "reject"
4. If all the 0s have now been crossed off, "accept". If any 0s remain, "reject".

Quiz 5.2: Based on the time complexity of the two-tape TM M_3, we can say that (pick the most accurate upper bound):
A) $A \in TIME(n^2)$
B) $A \in TIME(n \log n)$
C) $A \in TIME(n)$
D) $A \in TIME(\log n)$

* The complexity of language A depends on the model of computation selected.

In computability theory → the Church-Turing thesis implies that all reasonable models of computation are equivalent, i.e., they all decide the same languages.

In complexity theory → the choice of the model affects the time complexity of languages.

On a high-level: Across deterministic models, the time requirements don't differ too much. Thankfully, the classification system we will see is not sensitive to these small differences.

Complexity Relationships Among Models

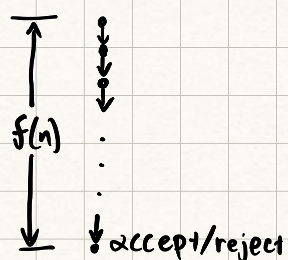
Theorem 7.8: Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time multitape Turing machine has an equivalent $O(t^2(n))$ time single-tape Turing machine.

Proof idea: Simulating each step of the multitape machine uses at most $O(t(n))$ steps on the single-tape machine. Hence, the total time is $O(t^2(n))$.

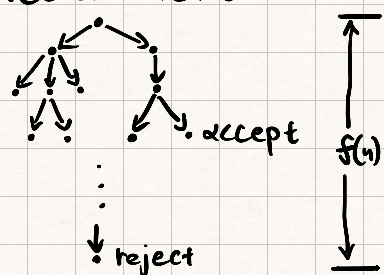
*At most polynomial time difference between time complexity measured in single-tape and multitape TM

Definition 7.9: Let M be a nondeterministic Turing machine that is a decider. The running time of M is the function $f: \mathbb{N} \rightarrow \mathbb{N}$, where $f(n)$ is the maximum number of steps that M uses on any branch of its computation on any input of length n.

Deterministic



Nondeterministic



Theorem 7.11: Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time nondeterministic single-tape Turing machine has an equivalent $2^{O(t(n))}$ time deterministic single-tape Turing machine.

*At most exponential time difference between time complexity measured in deterministic and nondeterministic TM.

The Class P (Sipser, Chapter 7.2)

All reasonable deterministic computational models are polynomially equivalent.

? Ignore polynomial differences: Recall that in asymptotic notation we disregarded constant factors. We now going to disregard much greater polynomial differences, i.e., between n and n^3. The reason is that we are interested in the polynomiality or nonpolynomiality of a problem.

Different Perspective

Definition 7.12: P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine. More formally

$$P = \bigcup_{k \geq 1} TIME(n^k)$$

- Interesting Observations
  1. P is invariant for all models of computation that are polynomially equivalent to the deterministic single-tape TM
  2. P contains the class of problems that realistically solvable in practice.

- We still use the angle-bracket notation $\langle \cdot \rangle$, to indicate a "reasonable" encoding to a string for a TM. Reasonable in a sense that we can encode/decode in polynomial time.

- A language $L \in P$, if there exists a TM M_L and a polynomial p' such that (1) $M_L(x)$ runs in time $p'(|x|)$, and (2) $x \in L$ if and only if $M_L(x) = 1$

- Class P is closed under composition. If algorithm A makes polynomially many calls to algorithm B and B runs in polynomial time, then A runs in polynomial time.

E.g., consider the language

$PATH = \{ \langle G, s, t \rangle \mid G$ is a directed graph that has a directed path from s to t$\}$

Theorem 7.14: $PATH \in P$

The Class NP (Sipser, Chapter 7.3)

In certain cases, attempts to avoid brute force haven't been succesful, i.e., we don't know any polynomial algorithm to solve them.

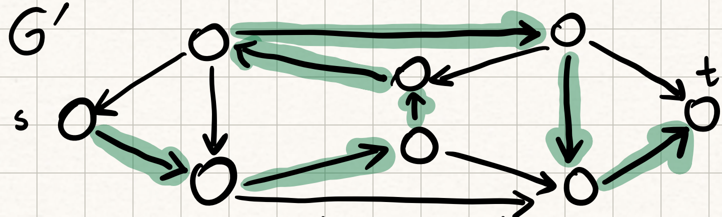
↳ why? Maybe because we haven't found the right algorithmic tools or maybe because these problems cannot be solved in polynomial time.

⚠ The complexities of many of these problems are linked! A polynomial time algorithm for one of these problems can be used to solve an entire class of problems.

Let's start with an example:

A Hamiltonian path in a directed graph G is a directed path that goes through each node exactly once. We can define the language

$HAMPATH = \{ \langle G, s, t \rangle \mid G$ is a directed graph with a Hamiltonian path from s to t$\}$



A Hamiltonian path

We can build a brute-force algorithm for testing if an input is a member of HAMPATH ⟹ slightly modify the brute-force approach for PATH.

- Interestingly, some problems, e.g., HAMPATH, have an interesting feature called polynomial verifiability. Even though we don't know of a fast way to construct their solution, we can efficiently convince someone of its existence by presenting it.

Definition 7.18: A verifier for a language L is an algorithm V, where
$L = \{x \mid V \text{ accepts } \langle x,w \rangle \text{ for some string } w\}$.

- A polynomial time verifier runs in polynomial time in the length of x.
A language L is polynomially verifiable if it has a polynomial time verifier.

- The "extra" string w is called a certificate, or witness, or proof, of membership in L.

- For example, for $\langle G', s, t \rangle$ the certificate is simply the green-colored path from s to t.

Definition 7.19: NP is the class of languages that have polynomial time verifiers

Alternative definition: $L \in NP$ if there exists a TM M_L (verifier) and a polynomial p' such that (1) $M_L(x,w)$ runs in time $p'(|x|)$, and (2) $x \in L$ if and only if there exists a $w \in \Sigma^*$ such that $M_L(x,w) = 1$.

Overall, L is in P if membership in P can be decided efficiently
L is in NP if membership in NP can be verified efficiently.

Theorem (not is Sipser): $P \subseteq NP \subseteq \bigcup_{c \geq 1} TIME(2^{n^c})$

$EXPTIME = \bigcup_c TIME(2^{n^c})$
alternative notation

Proof: The machine that solves/decides a language $L \in P$ can be used to generate the solution which can serve as a certificate w for a verifier which implies $L \in NP$. Therefore, $P \subseteq NP$.

Suppos $L \in NP$, then there exists a polynomial verifier V_L. Since V_L runs in polynomial time, it can read at most the first $p'(|x|)$ bits of w. Thus, w has length at most $p'(|x|)$.
To show that $L \in TIME(2^{n^c})$ we construct the following decider.

M = "On input x,
1. Run $M_L(x,w)$ for all strings $w \in \{0,1\}^{\leq p'(|x|)}$
2. If any of these results in $M_L(x,w)$ outputs "accept", then M outputs "accept". Otherwise, "reject". "

The running time of M on x is $O(2^{p'(|x|)} \cdot p'(|x|))$, thus, $L \in TIME(2^{n^c})$ for a constant c. Therefore, $NP \subseteq \bigcup_{c \geq 1} TIME(2^{n^c})$ ∎

The NP Class using nondeterministic TM

The following is a nondeterministic Turing machine (NTM) that decides HAMPATH in nondeterministic polynomial time.

M_1 = "On input $\langle G, s, t \rangle$, where G is a directed graph with nodes s,t:
1. Write a list of k numbers, $p_1, \dots, p_k$, where k is the number of nodes in G. Each number in the list is nondeterministically selected to be between 1 and k.
2. Check for repetitions in the list. If there is any, "reject"
3. Check if $s = p_1$ and $t = p_k$. If either fail, "reject".
4. For each $i \in [1, k-1]$, check if (p_i, p_{i+1}) is an edge of G. If any are not, "reject". Otherwise, "accept"

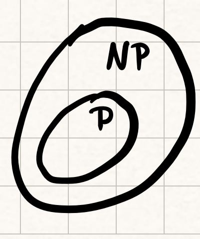
Theorem 7.20: A language is in NP if and only if it is decided by some nondeterministic polynomial time Turing machine

Definition 7.21:

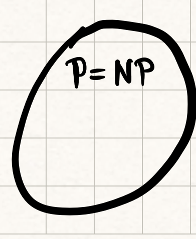
$$NTIME(t(n)) = \{L \mid L \text{ is a language decided by an } O(t(n)) \text{ time NTM}\}$$

Corollary 7.22: $NP = \bigcup_{c \geq 1} NTIME(n^c)$

Overall:
- The power of polynomial verifiability seems to be much greater than that of polynomial decidability.
- We are unable to prove the existence of a single language in NP that is not in P.

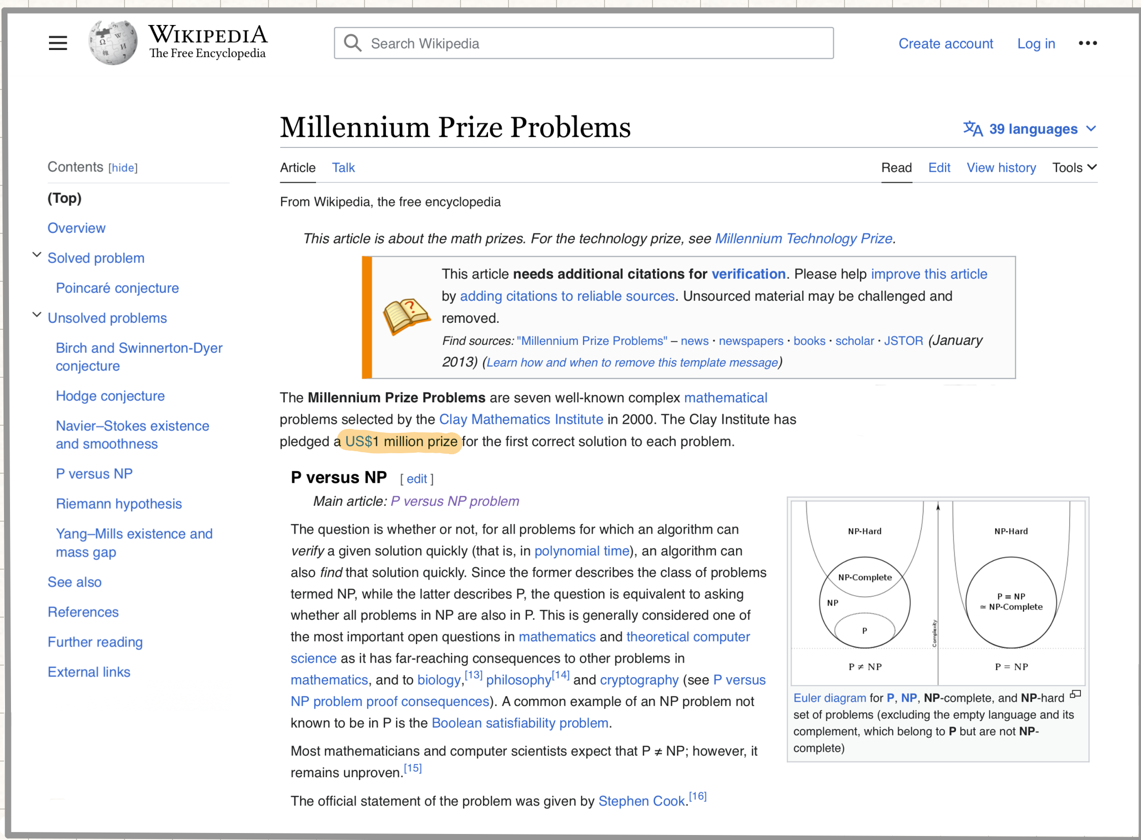


or



* The consensus is that the two classes are not equal. But we have no proof!

verifying a ingenious solution should be easier than constructing it on your own!



Wikipedia
The Free Encyclopedia

Search Wikipedia

Create account Log in

Millennium Prize Problems

39 languages

Article Talk Read Edit View history Tools

Contents [hide]

From Wikipedia, the free encyclopedia

This article is about the math prizes. For the technology prize, see Millennium Technology Prize.

This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed.
Find sources: "Millennium Prize Problems" – news · newspapers · books · scholar · JSTOR *(January 2013) (Learn how and when to remove this template message)*

The **Millennium Prize Problems** are seven well-known complex mathematical problems selected by the Clay Mathematics Institute in 2000. The Clay Institute has pledged a US$1 million prize for the first correct solution to each problem.

P versus NP [edit]

Main article: P versus NP problem

The question is whether or not, for all problems for which an algorithm can *verify* a given solution quickly (that is, in polynomial time), an algorithm can also *find* that solution quickly. Since the former describes the class of problems termed NP, while the latter describes P, the question is equivalent to asking whether all problems in NP are also in P. This is generally considered one of the most important open questions in mathematics and theoretical computer science as it has far-reaching consequences to other problems in mathematics, and to biology,[13] philosophy,[14] and cryptography (see P versus NP problem proof consequences). A common example of an NP problem not known to be in P is the Boolean satisfiability problem.

Euler diagram for **P**, **NP**, **NP**-complete, and **NP**-hard set of problems (excluding the empty language and its complement, which belong to **P** but are not **NP**-complete)

Most mathematicians and computer scientists expect that P ≠ NP; however, it remains unproven.[15]

The official statement of the problem was given by Stephen Cook.[16]

The best deterministic method currently known for deciding languages in NP uses exponential time, i.e.,

$$NP \subseteq EXPTIME$$