What are the differences between NP, NP-Complete and NP-Hard?

Asked 15 years ago Modified 1 month ago Viewed 756k times

1468

What are the differences between **NP**, **NP-Complete** and **NP-Hard**?

I am aware of many resources all over the web. I'd like to read your explanations, and the reason is they might be different from what's out there, or there is something that I'm not aware of.

computer-science complexity-theory np np-complete np-hard

Share

Improve this question

Follow

edited Jan 7, 2019 at 9:20

rez

2,067 ● 1 ● 22 ● 40

asked Dec 7, 2009 at 1:11

DarthVader

55k ● 78 ● 222 ● 306

13 Answers

Sorted by: Highest score (default)

1899

I assume that you are looking for intuitive definitions, since the technical definitions require quite some time to understand. First of all, let's remember a preliminary needed concept to understand those definitions.

- **Decision problem**: A problem with a **yes** or **no** answer.

Now, let us define those *complexity classes*.

P

P is a complexity class that represents the set of all decision problems that can be solved in polynomial time.

That is, given an instance of the problem, the answer yes or no can be decided in polynomial time.

Example

Given a connected graph `G`, can its vertices be coloured using two colours so that no edge is monochromatic?

Algorithm: start with an arbitrary vertex, color it red and all of its neighbours blue and continue. Stop when you run out of vertices or you are forced to make an edge have both of its endpoints be the same color.

NP

NP is a complexity class that represents the set of all decision problems for which the instances where the answer is "yes" have proofs that can be verified in polynomial time.

This means that if someone gives us an instance of the problem and a certificate (sometimes called a witness) to the answer being yes, we can check that it is correct in polynomial time.

Example

Integer factorisation is in NP. This is the problem that given integers `n` and `m`, is there an integer `f` with `1 < f < m`, such that `f` divides `n` (`f` is a small factor of `n`)?

This is a decision problem because the answers are yes or no. If someone hands us an instance of the problem (so they hand us integers `n` and `m`) and an integer `f` with `1 < f < m`, and claim that `f` is a factor of `n` (the certificate), we can check the answer in *polynomial time* by performing the division `n / f`.

NP-Complete

NP-Complete is a complexity class which represents the set of all problems `X` in NP for which it is possible to reduce any other NP problem `Y` to `X` in polynomial time.

Intuitively this means that we can solve `Y` quickly if we know how to solve `X` quickly. Precisely, `Y` is reducible to `X`, if there is a polynomial time algorithm `f` to transform instances `y` of `Y` to instances `x = f(y)` of `X` in polynomial time, with the property that the answer to `y` is yes, if and only if the answer to `f(y)` is yes.

Example

`3-SAT`. This is the problem wherein we are given a conjunction (ANDs) of 3-clause disjunctions (ORs), statements of the form

```
(x_v11 OR x_v21 OR x_v31) AND
(x_v12 OR x_v22 OR x_v32) AND
...                       AND
(x_v1n OR x_v2n OR x_v3n)
```

where each `x_vij` is a boolean variable or the negation of a variable from a finite predefined list `(x_1, x_2, ... x_n)`.

It can be shown that *every NP problem can be reduced to 3-SAT*. The proof of this is technical and requires use of the technical definition of NP (*based on non-deterministic Turing machines*). This is known as *Cook's theorem*.

What makes NP-complete problems important is that if a deterministic polynomial time algorithm can be found to solve one of them, every NP problem is solvable in polynomial time (one problem to rule them all).

NP-hard

Intuitively, these are the problems that are *at least as hard as the NP-complete problems*. Note that NP-hard problems *do not have to be in NP*, and *they do not have to be decision problems*.

The precise definition here is that *a problem* `X` *is NP-hard, if there is an NP-complete problem* `Y`*, such that* `Y` *is reducible to* `X` *in polynomial time*.

But since any NP-complete problem can be reduced to any other NP-complete problem in polynomial time, all NP-complete problems can be reduced to any NP-hard problem in polynomial time. Then, if there is a solution to one NP-hard problem in polynomial time, there is a solution to all NP problems in polynomial time.

Example

The halting problem is an NP-hard problem. This is the problem that given a program `P` and input `I`, will it halt? This is a decision problem but it is not in NP. It is clear that any NP-complete problem can be reduced to this one. As another example, any NP-complete problem is NP-hard.

My favorite NP-complete problem is the Minesweeper problem.

P = NP

This one is the most famous problem in computer science, and one of the most important outstanding questions in the mathematical sciences. In fact, the Clay Institute is offering one million dollars for a solution to the problem (Stephen Cook's writeup is quite good).

It's clear that P is a subset of NP. The open question is whether or not NP problems have deterministic polynomial time solutions. It is largely believed that they do not. Here is an outstanding recent article on the latest (and the importance) of the P = NP problem: The Status of the P versus NP problem.

The best book on the subject is Computers and Intractability by Garey and Johnson.

edited Nov 11 at 16:01

Arya

1,469 ● 2 ● 19 ● 44

answered Dec 7, 2009 at 1:46

jason

241k ● 35 ● 434 ● 530

48 @Paul Fisher: I'll show that SAT is reducible to the halting problem in polynomial time. Consider the following algorithm: given as input a proposition `I` over `n` variables, try all `2^n` possible assignments to the variables and halt if one satisfies the proposition and otherwise enter an infinite loop. We see that this algorithm halts if and only if `I` is satisfiable. Thus, if we had a polynomial time algorithm for solving the halting problem then we could solve SAT in polynomial time. Therefore, the halting problem is NP-hard. – jason Dec 7, 2009 at 3:52

6 @Jason - You can't reduce a decidable problem to an undecidable problem in that manner. Decidable problems have to result in a definitive yes or no answer in order to be considered to be decidable. The Halting Problem does not have a definitive yes or now answer since an arbitrary answer might throw any solution into a loop. – rjzii Dec 13, 2009 at 2:48

12 @Rob: Yes, I can. The definition of reducible does not require that the problem being reduced to be solvable. This is true for either many-one reductions or Turing reductions. – jason Dec 13, 2009 at 13:12

5 @Rob: Well, okay, if you want to continue this. First, "Decidable" is not synonomous with "decision problem" as you've used it. "Decidable" means, roughly, that there is an "effective method" for determining the answer. "Effective method", of course, has a technical definition. Moreover, "decidable" can also be defined in terms of "computable functions." So, the halting problem is a decision problem ("Does this program halt?" is a yes/no question) but it is undecidable; there is no effective method for determining whether or not an instance of the halting problem will halt. – jason Dec 13, 2009 at 23:16

32 Using Halting problem as a "classic example" of NP-hard problem is incorrect. This is like saying: "Pacific Ocean is a

classic example of a salt water aquarium." – Michael Mar 11, 2014 at 21:18

367

I've been looking around and seeing many long explanations. Here is a small chart that may be useful to summarise:

Notice how difficulty increases top to bottom: any *NP can be reduced to NP-Complete*, and any *NP-Complete can be reduced to NP-Hard*, all in P (polynomial) time.

If you can solve a more difficult class of problem in P time, that will mean you found how to solve all easier problems in P time (for example, proving P = NP, if you figure out how to solve any NP-Complete problem in P time).

```
_____________________________________________________________________
| Problem Type | Verifiable in P time | Solvable
in P time | Increasing Difficulty
_____________________________________________________________________
|
| P            |        Yes           |        Yes
|           |
| NP           |        Yes           |     Yes or
No *    |           |
| NP-Complete  |        Yes           |
Unknown       |           |
| NP-Hard      |     Yes or No **     |
Unknown ***   |           |
_____________________________________________________________________
V
```

Notes on `Yes` or `No` entries:

- * An NP problem that is also P is solvable in P time.
- ** An NP-Hard problem that is also NP-Complete is verifiable in P time.
- *** NP-Complete problems (all of which form a subset of NP-hard) might be. The rest of NP hard is not.

I also found this diagram quite useful in seeing how all these types correspond to each other (pay more attention to the left half of the diagram).

Share Improve this answer

Follow

edited Jul 23, 2015 at 14:35

nbro

16k ● 34 ● 119 ● 212

answered Oct 22, 2013 at 6:08

Johnson Wong

4,463 ● 1 ● 16 ● 6

3 I've a doubt related to your answer. I asked it in a separate question, but I was asked to post it here. Can you please help me here? stackoverflow.com/questions/21005651/… – Srikanth Jan 8, 2014 at 21:09

It is unknown whether NP-complete problems are solvable in polynomial time. Also, NP-complete problems are NP-hard, so some NP-hard problems are verifiable in polynomial time, and possible some also polynomial-time solvable. – Falk Hüffner Jan 9, 2014 at 16:50

1 This table is incorrect and self-contradictory. Even if you assume that NP!=P, which has not been proved yet, it would still be incorrect. For example, NP-Hard class includes NP-Complete problems; therefore your table claims that NP-

Complete problems are simultaneously verifiable in polynomial time and not verifiable in polynomial time.
– Michael Feb 21, 2014 at 18:11

4 @FalkHüffner Thanks, table is updated (was an error in translating from the Venn diagram). – Johnson Wong Feb 26, 2014 at 7:16

1 @PeterRaeves All NP-complete problems are NP-hard, by definition: NP-complete = (NP and NP-hard). The inverse is not true: there are problems (such as the Halting Problem) in NP-hard that are not in NP-complete. "NP (not solvable in polynomial time) " -- that's not what NP means. NP is "Non-deterministic-polynomial". All problems in P are also in NP. Whether the inverse is true is famously unknown.
– Jim Balter Jun 13, 2015 at 9:22

168

P (Polynomial Time): As name itself suggests, these are the problems which can be solved in polynomial time.

NP (Non-deterministic-polynomial Time): These are the decision problems which can be verified in polynomial time. That means, if I claim that there is a polynomial time solution for a particular problem, you ask me to prove it. Then, I will give you a proof which you can easily verify in polynomial time. These kind of problems are called NP problems. Note that, here we are not talking about whether there is a polynomial time solution for this problem or not. But we are talking about verifying the solution to a given problem in polynomial time.

NP-Hard: These are at least as hard as the hardest problems in NP. If we can solve these problems in

polynomial time, we can solve any NP problem that can possibly exist. Note that these problems are not necessarily NP problems. That means, we may/may-not verify the solution to these problems in polynomial time.

NP-Complete: These are the problems which are both NP and NP-Hard. That means, if we can solve these problems, we can solve any other NP problem and the solutions to these problems can be verified in polynomial time.

Share Improve this answer

Follow

edited Jun 25, 2020 at 7:09

Hossein Narimani Rad

32.4k ● 18 ● 91 ● 121

answered Oct 4, 2013 at 3:50

Nagakishore Sidde

2,737 ● 1 ● 18 ● 10

21 Best answer as it's short, uses just enough terminology, has normal human sentences (not the hard to read let's-be-as-correct-as-possible stuff), and surprisingly enough is the only answer that writes what N stands for. – meaning-matters Aug 1, 2018 at 14:01

4 This answer is better than the others but I would explain NP as "Problems which can be solved in polynomial time when running on a **non-deterministic** Turing machine", where as P problems are "Problems which can be solved in polynomial time when running on a **deterministic** Turing machine". That's really where the "non-deterministic" in NP comes from. – Noah Nuebling Jul 20, 2022 at 22:51

This is a very informal answer to the question asked.

Can 3233 be written as the product of two other numbers bigger than 1? Is there any way to walk a path around all of the Seven Bridges of Königsberg without taking any bridge twice? These are examples of questions that share a common trait. It may not be obvious how to efficiently determine the answer, but if the answer is 'yes', then there's a short and quick to check proof. In the first case a non-trivial factorization of 61 (53 being the other prime factor); in the second, a route for walking the bridges (fitting the constraints).

A **decision problem** is a collection of questions with yes or no answers that vary only in one parameter. Say the problem COMPOSITE={"Is `n` composite": `n` is an integer} or EULERPATH={"Does the graph `G` have an Euler path?": `G` is a finite graph}.

Now, some decision problems lend themselves to efficient, if not obvious algorithms. Euler discovered an efficient algorithm for problems like the "Seven Bridges of Königsberg" over 250 years ago.

On the other hand, for many decision problems, it's not obvious how to get the answer -- but if you know some additional piece of information, it's obvious how to go about proving you've got the answer right. COMPOSITE is like this: Trial division is the obvious algorithm, and it's slow: to factor a 10 digit number, you have to try something like 100,000 possible divisors. But if, for

example, somebody told you that 61 is a divisor of 3233, simple long division is a efficient way to see that they're correct.

The complexity class **NP** is the class of decision problems where the 'yes' answers have short to state, quick to check proofs. Like COMPOSITE. One important point is that this definition doesn't say anything about how hard the problem is. If you have a correct, efficient way to solve a decision problem, just writing down the steps in the solution is proof enough.

Algorithms research continues, and new clever algorithms are created all the time. A problem you might not know how to solve efficiently today may turn out to have an efficient (if not obvious) solution tomorrow. In fact, it took researchers until 2002 to find an efficient solution to COMPOSITE! With all these advances, one really has to wonder: Is this bit about having short proofs just an illusion? Maybe *every* decision problem that lends itself to efficient proofs has an efficient solution? Nobody knows.

Perhaps the biggest contribution to this field came with the discovery a peculiar class of NP problems. By playing around with circuit models for computation, Stephen Cook found a decision problem of the NP variety that was provably as hard or harder than *every* other NP problem. An efficient solution for the boolean satisfiability problem could be used to create an efficient solution to *any other* problem in NP. Soon after, Richard Karp showed that a

number of other decision problems could serve the same purpose. These problems, in a sense the "hardest" problems in NP, became known as **NP-complete** problems.

Of course, NP is only a class of decision problems. Many problems aren't naturally stated in this manner: "find the factors of N", "find the shortest path in the graph G that visits every vertex", "give a set of variable assignments that makes the following boolean expression true". Though one may informally talk about some such problems being "in NP", technically that doesn't make much sense -- they're not decision problems. Some of these problems might even have the same sort of power as an NP-complete problem: an efficient solution to these (non-decision) problems would lead directly to an efficient solution to any NP problem. A problem like this is called **NP-hard**.

Share Improve this answer

Follow

edited Jan 23, 2022 at 5:04

Community Bot

1 ● 1

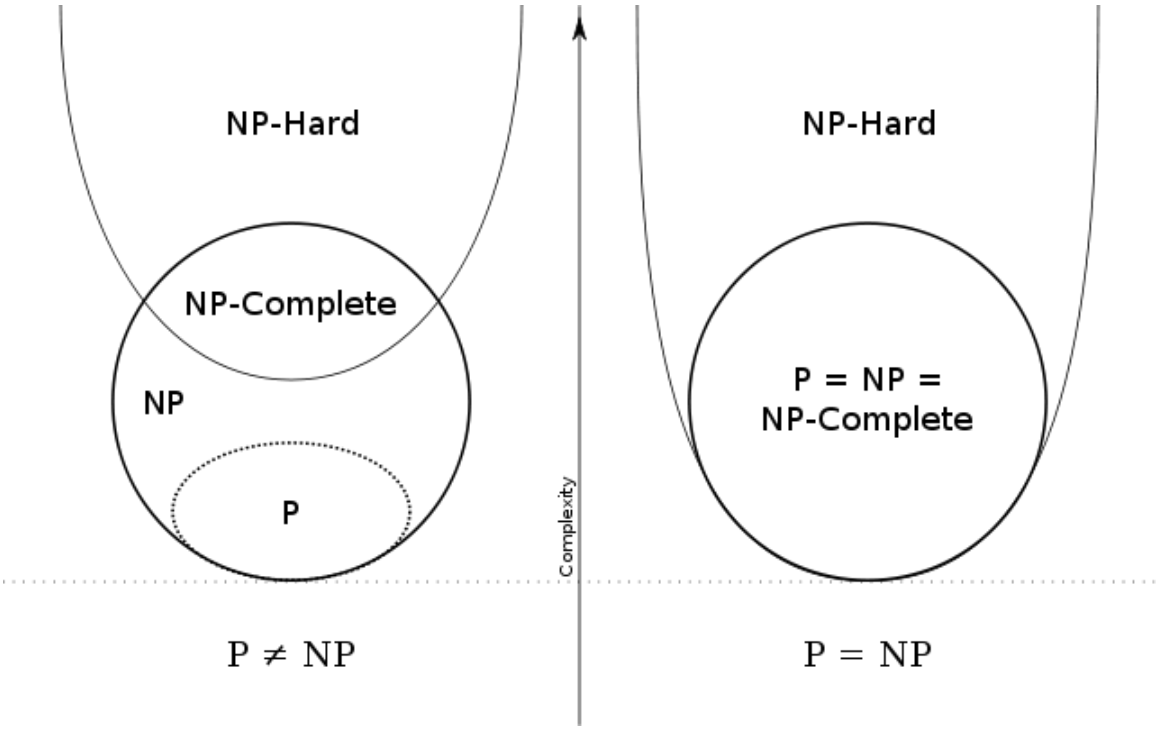
answered Dec 7, 2009 at 2:42

Managu

9,039 ● 2 ● 31 ● 36

80

In addition to the other great answers, here is the typical schema people use to show the difference between NP, NP-Complete, and NP-Hard:



Share Improve this answer

Follow

answered Nov 24, 2014 at 17:50

Franck Dernoncourt

82.8k ● 77 ● 364 ● 539

3 Is it proven that there exist a problem in NP-Hard that is not in NP-Complete? Because this image suggests it. Thank you. – Hilder Vitor Lima Pereira Dec 4, 2014 at 21:23

11 @VitorLima yes e.g. EXPSPACE-complete problems are NP-hard but proven not to be NP-complete.
– Franck Dernoncourt Dec 4, 2014 at 21:43

2 Ok, thank you. I found some references talking about it. For example, this one: princeton.edu/~achaney/tmve/wiki100k/docs/NP-hard.html
– Hilder Vitor Lima Pereira Dec 4, 2014 at 23:28

57

The easiest way to explain P v. NP and such without getting into technicalities is to compare "word problems" with "multiple choice problems".

When you are trying to solve a "word problem" you have to find the solution from scratch. When you are trying to solve a "multiple choice problems" you have a choice: either solve it as you would a "word problem", or try to plug in each of the answers given to you, and pick the candidate answer that fits.

It often happens that a "multiple choice problem" is much easier than the corresponding "word problem": substituting the candidate answers and checking whether they fit may require significantly less effort than finding the right answer from scratch.

Now, if we would agree the effort that takes polynomial time "easy" then the class P would consist of "easy word problems", and the class NP would consist of "easy multiple choice problems".

The essence of P v. NP is the question: "Are there any easy multiple choice problems that are not easy as word problems"? That is, are there problems for which it's easy to verify the validity of a given answer but finding that answer from scratch is difficult?

Now that we understand intuitively what NP is, we have to challenge our intuition. It turns out that there are "multiple choice problems" that, in some sense, are hardest of them all: if one would find a solution to one of those "hardest of them all" problems one would be able to find a solution to ALL NP problems! When Cook discovered this 40 years ago it came as a complete surprise. These "hardest of them all" problems are known as NP-hard. If

you find a "word problem solution" to one of them you would automatically find a "word problem solution" to each and every "easy multiple choice problem"!

Finally, NP-complete problems are those that are simultaneously NP and NP-hard. Following our analogy, they are simultaneously "easy as multiple choice problems" and "the hardest of them all as word problems".

Share Improve this answer

Follow

answered Aug 8, 2013 at 20:41

Michael

5,879 ● 3 ● 37 ● 61

21

NP-complete problems are those problems that are both NP-Hard and in the complexity class NP. Therefore, to show that any given problem is NP-complete, you need to show that the problem is both in NP and that it is NP-hard.

Problems that are in the NP complexity class can be solved non-deterministically in polynomial time and a possible solution (i.e., a certificate) for a problem in NP can be verified for correctness in polynomial time.

An example of a non-deterministic solution to the k-clique problem would be something like:

1) randomly select k nodes from a graph

2) verify that these k nodes form a clique.

The above strategy is polynomial in the size of the input graph and therefore the k-clique problem is in NP.

Note that all problems deterministically solvable in polynomial time are also in NP.

Showing that a problem is NP-hard typically involves a reduction from some other NP-hard problem to your problem using a polynomial time mapping: http://en.wikipedia.org/wiki/Reduction_(complexity)

Share Improve this answer Follow

answered Dec 7, 2009 at 1:45

awesomo
8,806 ● 2 ● 23 ● 24

4 Not that I see anything in this answer that is incorrect, but I don't know why it was accepted. It doesn't really offer much to what the OP was asking. It's not really even different than the standard explanations of these problems, and there aren't any clear explanations as to what makes these problems in these classes. Not worth a downvote, but certainly not worth answer acceptance. – user124493 Dec 7, 2009 at 1:50

19

I think we can answer it much more succinctly. I answered a related question, and copying my answer from there

But first, an NP-hard problem is a problem for which we cannot prove that a polynomial time solution exists. NP-hardness of some "problem-P" is usually proven by converting an already proven NP-hard problem to the "problem-P" in polynomial time.

To answer the rest of question, you first need to understand which NP-hard problems are also NP-complete. If an NP-hard problem belongs to set NP, then it is NP-complete. To belong to set NP, a problem needs to be

(i) a decision problem,
(ii) the number of solutions to the problem should be finite and each solution should be of polynomial length, and
(iii) given a polynomial length solution, we should be able to say whether the answer to the problem is yes/no

Now, it is easy to see that there could be many NP-hard problems that do not belong to set NP and are harder to solve. As an intuitive example, the optimization-version of traveling salesman where we need to find an actual schedule is harder than the decision-version of traveling salesman where we just need to determine whether a schedule with length <= k exists or not.

Share Improve this answer

Follow

edited May 23, 2017 at 11:55

Community Bot

1 • 1

answered Jun 18, 2011 at 16:52

Sushant Sharma

447 • 4 • 8

5

There are really nice answers for this particular question, so there is no point to write my own explanation. So I will try to contribute with an excellent resource about different classes of computational complexity.

For someone who thinks that computational complexity is only about P and NP, here is the most exhaustive resource about different computational complexity problems. Apart from problems asked by OP, it listed approximately 500 different classes of computational problems with nice descriptions and also the list of fundamental research papers which describe the class.

Share Improve this answer

Follow

answered Jan 14, 2014 at 1:56

Salvador Dali

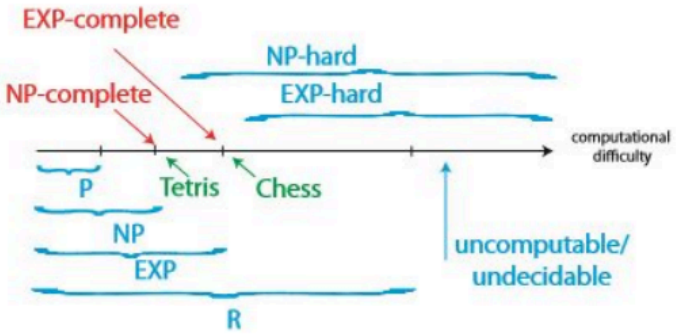
222k • 151 • 723 • 762

> Thanks for that. The old link is broken, though. The new address of that site is complexityzoo.net/Complexity_Zoo – Alexandre M Jun 9, 2022 at 0:01

Find some interesting defintion:

4

If P ! = NP



Definitions:

· P = {problems solvable in polynomial (nc) time} (what this class is all about)

· EXP = {problems solvable in exponential (2nc) time}

· R = {problems solvable in finite time} "recursive" [Turing 1936; Church 1941]

· NP ={Decision problems solvable in polynomial time via a "lucky" algorithm}.

In other words, NP ={decision problems with solutions that can be "checked" in polynomial time}.

· NP-hard = "as hard as" every problem ∈ NP. In fact NP-complete = NP ∩ NPhard.

Share Improve this answer

Follow

answered Jun 11, 2019 at 3:06

sendon1982

11.2k ● 68 ● 49

1 What is the source for the image?
– Gaslight Deceive Subvert Apr 7, 2022 at 16:58

2

As I understand it, an *np-hard* problem is not "harder" than an *np-complete* problem. In fact, by definition, every np-complete problem is:

1. in NP
2. np-hard

> We can now define the set of NP-complete languages, which are the hardest problems in NP.
>
> A language $L \subseteq \{0,1\}^*$ is ***NP-complete*** if
>
> 1. $L \in$ NP, and
> 2. $L' \leq_P L$ for every $L' \in$ NP.
>
> If a language L satisfies property 2, but not necessarily property 1, we say that L is ***NP-hard***.
>
> -- Intro. to Algorithms (3ed) by Cormen, Leiserson, Rivest, and Stein, pg 1069

Condition 1. and 2. translated to English:

1. Language L is in NP, and
2. Every NP language is polynomial time reducible to language L.

Share Improve this answer

Follow

edited May 12, 2021 at 11:56

Shaan K

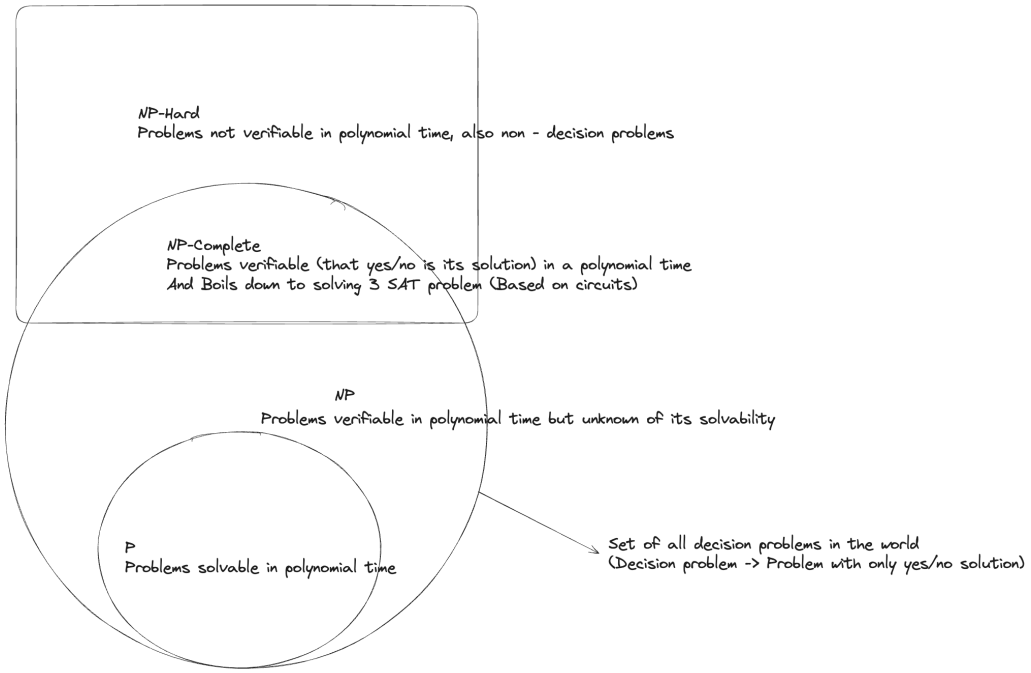
376 ● 1 ● 7

answered Aug 13, 2014 at 13:57

MrDrews

2,137 ● 2 ● 22 ● 22

9 Your understanding is incorrect. Your definition of NP-complete is correct but has no bearing on your first statement. All problems in NP-hard are *at least as hard* as those in NP-complete; some (e.g. the Halting Problem, which is infinitely hard, and en.wikipedia.org/wiki/EXPSPACE) are provably harder. – Jim Balter Jun 13, 2015 at 9:30



Created this diagram for better understanding: Let me know if any point is wrong or missed here.

Share Improve this answer

Follow

answered Apr 13 at 19:09

YASDE

31 ● 4

Your answer could be improved with additional supporting information. Please edit to add further details, such as citations or documentation, so that others can confirm that your answer is correct. You can find more information on how to write good answers in the help center. – Community Bot Apr 27 at 13:02

P (Polynomial-time)

Set of decisional problems that can be **resolved in polynomial time**.

NP (Non-deterministic Polynomial-time)

Set of decisional problems that can be **verified in polynomial time**.

Lemma : $P \subseteq NP$

A problem (in *P*) that can be resolved in polynomial time with an algorithm, can also be verified in polynomial time with the same algorithm (therefore it is in *NP*).

note: $P \subseteq NP$ reads as : " *P* is a subset of *NP* " and it means that either $P = NP$ or $P \subset NP$.

P versus NP problem

It is an unresolved problem with a prize of 1'000'000 $ the proof that either:

- **P = NP** The set *P* is equal to the set *NP*.
- **P ⊂ NP** The set *P* is a proper subset of the set *NP*. (which implies that **P ≠ NP**).

NP-complete

Set of decisional problems. Proper subset of *NP* (can always be verified in polynomial time).

NP-complete ⊂ NP

Has the following characteristics:

- Every problem in *NP* can be **reduced in polynomial time** to a problem in *NP-Complete*.
- If an algorithm capable of solving any *NP-complete* problem were found then such an algorithm would be **capable of solving any problem in *NP***. Such an algorithm has not yet been found.

Example of *NP-complete* problem: SAT

NP-hard

Set of problems not limited to decisional problems; at least as hard as the problems in *NP-complete*.

NP-hard is a proper superset of *NP-complete*. (may or may not be verifiable in polynomial time).

NP-complete ⊂ NP-hard

Has the following characteristics:

- Every problem in *NP-complete* can be **reduced in polynomial time** to a problem in *NP-Hard*.
- If an algorithm capable of solving any *NP-hard* problem were found then such an algorithm would be **capable of solving any problem in *NP***. Such an algorithm has not yet been found.

Example of *NP-hard* problem: SAT (since *NP-Complete* ⊂ *NP-Hard*)

Example of *NP-Hard* but not *NP-Complete* problem: Halting Problem

Share Improve this answer Follow

edited Jan 28 at 13:20

answered Jul 6, 2023 at 7:54

Marco D.G.
2,405 ● 20 ● 32

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.