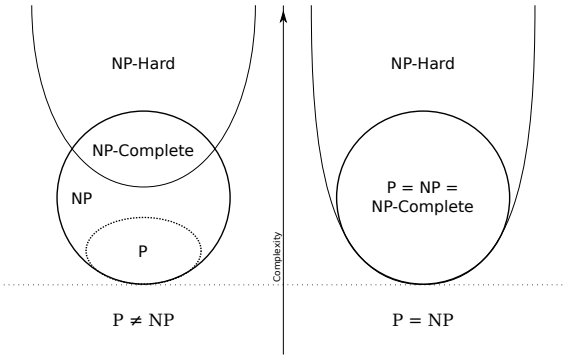
NP (complexity)



*Euler diagram for P, NP, NP-complete, and NP-hard set of problems. Under the assumption that P≠NP, the existence of problems within **NP** but outside both **P** and **NP**-complete was established by Ladner.*[1]

In computational complexity theory, **NP** is a complexity class used to describe certain types of decision problems. Informally, **NP** is the set of all decision problems for which the instances where the answer is "yes" have efficiently *verifiable* proofs. More precisely, these proofs have to be verifiable by deterministic computations that can be performed in polynomial time.

Equivalently, the formal definition of **NP** is the set of decision problems *solvable* in polynomial time by a theoretical non-deterministic Turing machine. This second definition is the basis for the abbreviation **NP**, which stands for "nondeterministic, polynomial time." However, the *verifier-based definition* tends to be more intuitive and practical in common applications compared to the formal *machine definition*. The two definitions are equivalent because the algorithm for the machine definition consists of two phases, the first of which consists of a guess about the solution, which is generated in a non-deterministic way, while the second phase consists of a deterministic algorithm that verifies or rejects the guess as a valid solution to the problem.[2]

A method for solving a decision problem is given in the form of an algorithm. In the above definitions for **NP**, *polynomial time* refers to the increasing number of machine operations needed by an algorithm relative to the size of the problem. Polynomial time is therefore a measure of efficiency of an algorithm. Decision problems are commonly categorized into complexity classes (such as **NP**) based on the fastest known machine algorithms. As such, decision problems may change class if a faster algorithm is discovered.

The complexity class **P** is contained in **NP**, but **NP** contains many important problems, the hardest of which are called **NP**-complete problems, whose solutions are sufficient to deal with any other NP problem in polynomial time. The most important open question in complexity theory, the **P** versus **NP** ("P=NP") problem, asks whether polynomial time algorithms actually exist for solving **NP**-complete, and by corollary, all **NP** problems. It is widely believed that this is not the case.[3]

The complexity class **NP** (which have efficiently verifiable proofs where the answer is "yes") is also related to the complexity class **co-NP** (which have efficiently verifiable proofs where the answer is "no"). Whether or not **NP = co-NP** is another outstanding question in complexity theory.

1 Formal definition

The complexity class **NP** can be defined in terms of NTIME as follows:

$$\mathrm{NP} = \bigcup_{k \in \mathbb{N}} \mathrm{NTIME}(n^k).$$

where $\mathrm{NTIME}(n^k)$ is the set of decision problems that can be solved by a non-deterministic Turing machine in $O(n^k)$ time.

Alternatively, **NP** can be defined using deterministic Turing machines as verifiers. A language L is in **NP** if and only if there exist polynomials p and q, and a deterministic Turing machine M, such that

- For all x and y, the machine M runs in time $p(|x|)$ on input (x,y)
- For all x in L, there exists a string y of length $q(|x|)$ such that $M(x,y) = 1$
- For all x not in L and all strings y of length $q(|x|)$, $M(x,y) = 0$

2 Introduction

Many natural computer science problems are covered by the class **NP**. In particular, the decision versions of many interesting search problems and optimization problems are contained in **NP**.

2.1 Verifier-based definition

In order to explain the verifier-based definition of **NP**, let us consider the subset sum problem: Assume that we are given some integers, such as $\{-7, -3, -2, 5, 8\}$, and we wish to know whether some of these integers sum up to zero. In this example, the answer is "yes", since the subset of integers $\{-3, -2, 5\}$ corresponds to the sum $(-3) + (-2) + 5 = 0$. The task of deciding whether such a subset with sum zero exists is called the *subset sum problem*.

To answer if some of the integers add to zero we can create an algorithm which obtains all the possible subsets. As the number of integers that we feed into the algorithm becomes larger, the number of subsets grows exponentially and so does the computation time. However, notice that, if we are given a particular subset (often called a certificate), we can easily check or *verify* whether the subset sum is zero, by just summing up the integers of the subset. So if the sum is indeed zero, that particular subset is the *proof* or witness for the fact that the answer is "yes". An algorithm that verifies whether a given subset has sum zero is called *verifier*.

More generally, a problem is said to be in **NP** if there exists a verifier V for the problem. Given any instance I of problem P, where the answer is "yes", there must exist a certificate (also called a witness) W such that, given the ordered pair (I,W) as input, V returns the answer "yes" in polynomial time. Furthermore, if the answer to I is "no", the verifier will return "no" with input (I,W) for all possible W. Note that V could return the answer "No" even if the answer to I is "yes", if W is not a valid witness. For example, in the subset sum problem, if there is a subset whose sum is zero, but we select W to be a subset whose sum is not zero, the verifier will return "No", while if there is no subset whose sum is zero, the verifier will return "no" regardless of the choice of W. The verifier needs only polynomial time (it just needs to check whether W is really a subset of I, and whether the sum of W is zero), so the subset sum problem is in **NP**.

The "no"-answer version of this problem is stated as: "given a finite set of integers, does every non-empty subset have a nonzero sum?". The verifier-based definition of **NP** does *not* require an easy-to-verify certificate for the "no"-answers. The class of problems with such certificates for the "no"-answers is called **co-NP**. In fact, it is an open question whether all problems in **NP** also have certificates for the "no"-answers and thus are in **co-NP**.

2.2 Machine-definition

Equivalent to the verifier-based definition is the following characterization: **NP** is the set of decision problems solvable by a non-deterministic Turing machine that runs in polynomial time. (This means that there is an accepting computation path if a word is in the language – **co-NP** is defined dually with rejecting paths.) This definition is equivalent to the verifier-based definition because a non-deterministic Turing machine could solve an NP problem in polynomial time by non-deterministically selecting a certificate and running the verifier on the certificate. Similarly, if such a machine exists, then a polynomial time verifier can naturally be constructed from it.

2.3 Examples

This is an incomplete list of problems that are in **NP**.

- All problems in **P** (For, given a certificate for a problem in **P**, we can ignore the certificate and just solve the problem in polynomial time. Alternatively, a deterministic Turing machine is also trivially a non-deterministic Turing machine that just happens to not use any non-determinism.)
- The decision problem version of the integer factorization problem: given integers n and k, is there a factor f with $1 < f < k$ and f dividing n?
- The graph isomorphism problem of determining whether two graphs can be drawn identically
- All NP-complete problems, e.g.:
  - A variant of the travelling salesman problem, where we want to know if there is a route of some length that goes through all the nodes in a certain network
  - The boolean satisfiability problem, where we want to know whether or not a certain formula in propositional logic with boolean variables is true for some value of the variables

3 Properties

NP is closed under:

- union
- intersection
- concatenation
- Kleene star
- reversal

It is not known whether NP is closed under complement (this question so-called "NP versus co-NP" question)

4 Why some NP problems are hard to solve

Because of the many important problems in this class, there have been extensive efforts to find polynomial-time algorithms for problems in **NP**. However, there remain a large number of problems in **NP** that defy such attempts, seeming to require super-polynomial time. Whether these problems are not decidable in polynomial time is one of the greatest open questions in computer science (see **P** versus **NP** ("P=NP") problem for an in-depth discussion).

An important notion in this context is the set of NP-complete decision problems, which is a subset of **NP** and might be informally described as the "hardest" problems in **NP**. If there is a polynomial-time algorithm for even *one* of them, then there is a polynomial-time algorithm for *all* the problems in **NP**. Because of this, and because dedicated research has failed to find a polynomial algorithm for any **NP**-complete problem, once a problem has been proven to be **NP**-complete this is widely regarded as a sign that a polynomial algorithm for this problem is unlikely to exist.

However, in practical uses, instead of spending computational resources looking for an optimal solution, a good enough (but potentially suboptimal) solution may often be found in polynomial time. Also, the real life applications of some problems are easier than their theoretical equivalents.

5 Equivalence of definitions

The two definitions of **NP** as the class of problems solvable by a nondeterministic Turing machine (TM) in polynomial time and the class of problems verifiable by a deterministic Turing machine in polynomial time are equivalent. The proof is described by many textbooks, for example Sipser's *Introduction to the Theory of Computation*, section 7.3.

To show this, first suppose we have a deterministic verifier. A nondeterministic machine can simply nondeterministically run the verifier on all possible proof strings (this requires only polynomially many steps because it can nondeterministically choose the next character in the proof string in each step, and the length of the proof string must be polynomially bounded). If any proof is valid, some path will accept; if no proof is valid, the string is not in the language and it will reject.

Conversely, suppose we have a nondeterministic TM called A accepting a given language L. At each of its polynomially many steps, the machine's computation tree branches in at most a finite number of directions. There must be at least one accepting path, and the string describing this path is the proof supplied to the verifier. The verifier can then deterministically simulate A, following only the accepting path, and verifying that it accepts at the end. If A rejects the input, there is no accepting path, and the verifier will always reject.

6 Relationship to other classes

NP contains all problems in **P**, since one can verify any instance of the problem by simply ignoring the proof and solving it. **NP** is contained in **PSPACE**—to show this, it suffices to construct a **PSPACE** machine that loops over all proof strings and feeds each one to a polynomial-time verifier. Since a polynomial-time machine can only read polynomially many bits, it cannot use more than polynomial space, nor can it read a proof string occupying more than polynomial space (so we do not have to consider proofs longer than this). **NP** is also contained in EXPTIME, since the same algorithm operates in exponential time.

co-NP contains those problems which have a simple proof for *no* instances, sometimes called counterexamples. For example, primality testing trivially lies in co-NP, since one can refute the primality of an integer by merely supplying a nontrivial factor. **NP** and co-NP together form the first level in the polynomial hierarchy, higher only than **P**.

NP is defined using only deterministic machines. If we permit the verifier to be probabilistic (this however, is not necessarily a BPP machine[4]), we get the class **MA** solvable using an Arthur-Merlin protocol with no communication from Arthur to Merlin.

NP is a class of decision problems; the analogous class of function problems is FNP.

The only known strict inclusions came from the time hierarchy theorem and the space hierarchy theorem, and respectively they are **NP** $\subsetneq$ **NEXPTIME** and **NP** $\subsetneq$ **EXPSPACE**.

7 Other characterizations

In terms of descriptive complexity theory, **NP** corresponds precisely to the set of languages definable by existential second-order logic (Fagin's theorem).

NP can be seen as a very simple type of interactive proof system, where the prover comes up with the proof certificate and the verifier is a deterministic polynomial-time machine that checks it. It is complete because the right proof string will make it accept if there is one, and it is sound because the verifier cannot accept if there is no acceptable proof string.

A major result of complexity theory is that **NP** can be characterized as the problems solvable by probabilistically checkable proofs where the verifier uses

O(log n) random bits and examines only a constant number of bits of the proof string (the class **PCP**(log n, 1)). More informally, this means that the **NP** verifier described above can be replaced with one that just "spot-checks" a few places in the proof string, and using a limited number of coin flips can determine the correct answer with high probability. This allows several results about the hardness of approximation algorithms to be proven.

8 Example

The decision version of the travelling salesman problem is in **NP**. Given an input matrix of distances between n cities, the problem is to determine if there is a route visiting all cities with total distance less than k.

A proof certificate can simply be a list of the cities. Then verification can clearly be done in polynomial time by a deterministic Turing machine. It simply adds the matrix entries corresponding to the paths between the cities.

A non-deterministic Turing machine can find such a route as follows:

- At each city it visits it "guesses" the next city to visit, until it has visited every vertex. If it gets stuck, it stops immediately.
- At the end it verifies that the route it has taken has cost less than k in O(n) time.

One can think of each guess as "forking" a new copy of the Turing machine to follow each of the possible paths forward, and if at least one machine finds a route of distance less than k, that machine accepts the input. (Equivalently, this can be thought of as a single Turing machine that always guesses correctly)

A binary search on the range of possible distances can convert the decision version of Traveling Salesman to the optimization version, by calling the decision version repeatedly (a polynomial number of times).

9 See also

- Turing machine

10 References

[1] R. E. Ladner "On the structure of polynomial time reducibility," J.ACM, 22, pp. 151–171, 1975. Corollary 1.1. ACM site.

[2] Alsuwaiyel, M. H.: *Algorithms: Design Techniques and Analysis*, p. 283

[3] William Gasarch (June 2002). "The P=?NP poll" (PDF). *SIGACT News*. **33** (2): 34–47. doi:10.1145/1052796.1052804. Retrieved 2008-12-29.

[4] https://complexityzoo.uwaterloo.ca/Complexity_Zoo:E#existsbpp

General

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 34.2: Polynomial-time verification, pp. 979–983.
- Michael Sipser (1997). *Introduction to the Theory of Computation*. PWS Publishing. ISBN 0-534-94728-X. Sections 7.3–7.5 (The Class NP, NP-completeness, Additional NP-complete Problems), pp. 241–271.
- David Harel, Yishai Feldman. Algorithmics: The Spirit of Computing, Addison-Wesley, Reading, MA, 3rd edition, 2004.

11 External links

- *Complexity Zoo*: NP
- Graph of NP-complete Problems
- American Scientist primer on traditional and recent complexity theory research: "Accidental Algorithms"

12 Text and image sources, contributors, and licenses

12.1 Text

- **NP (complexity)** *Source:* https://en.wikipedia.org/wiki/NP_(complexity)?oldid=762373800 *Contributors:* AxelBoldt, Jan Hidders, Eijkhout, Perry Bebbington, Arvindn, Michael Hardy, Nixdorf, Zeno Gantner, Docu, Whkoh, Poor Yorick, Timwi, Dcoetzee, Furrykef, Head, Robbot, Falcone~enwiki, Wikibot, Tea2min, Giftlite, Jao, Mellum, Andris, Prosfilaes, Gdr, Creidieki, Mormegil, Blokhead, Helohe, Ascánder, Rnsanchez, Dmr2, DcoetzeeBot~enwiki, EmilJ, Longhair, Giraffedata, Obradovic Goran, Cheezycrust, Jonas.b.jensen, Shreevatsa, Oliphaunt, Jacobolus, Ruud Koot, Qwertyus, AndyKali, AySz88, Marozols, OmriSegal, FlaBot, Nihiltres, GünniX, Tylerl, Chobot, YurikBot, Wavelength, Pi Delport, Ogai, Trovatore, Wap, Kkmurray, Gulliveig, Josh3580, Tsiaojian lee, That Guy, From That Show!, SmackBot, Wdr1, Jcarroll, Vvarkey, Kaiserb, Duncancumming, Christan80, Spiritia, Ajnosek, Hypnosifl, Kissedsmiley, Ylloh, Clecio~enwiki, Thijs!bot, Qwyrxian, Electron9, Widefox, Weixifan, Oddity-, GromXXVII, Ahmed saeed, Catslash, TehKeg, R'n'B, HPRappaport, Avkulkarni, Vegasprof, Tarotcards, VolkovBot, LokiClock, Wombleme, Kinkydarkbird, Veganfanatic, AlleborgoBot, DevUrandom, Flyer22 Reborn, Paolo.dL, Allmightyduck, Fuddle, Kotniski, William Ortiz, Bender2k14, Mikaey, DumZiBoT, Addbot, Roentgenium111, MrVanBot, CarsracBot, Abovechief, Luckas-bot, Yobot, AnomieBOT, Rubinbot, Citation bot, Twri, Ikemccaslin, NavyFlyer1325, Miym, Nippashish, JonDePlume, FrescoBot, Chenopodiaceous, Albertzeyer, WuTheFWasThat, RobinK, Jasonwlh314, JumpDiscont, ParotWise, EmausBot, John of Reading, WikitanvirBot, Wikipelli, Slawekb, ZéroBot, Darthhappyface, Zephyrus Tavvier, Mentibot, LMolr, Snotbot, Widr, Zakhalesh, BG19bot, Filip13041982, Durron597, Silvercast234, Comp.arch, Blackbombchu, Elektrolurch, Monkbot, Thibaut120094, JonJacobsen, Wikiedit738, Whikie, P3d4r0, Scrabbler94, Gasarch, ElnaserAbdelwahab, Bender the Bot and Anonymous: 119

12.2 Images

- **File:P_np_np-complete_np-hard.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/a/a0/P_np_np-complete_np-hard.svg *License:* CC BY-SA 3.0 *Contributors:* Original. *Original artist:* Behnam Esfahbod
- **File:Question_dropshade.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/d/dd/Question_dropshade.png *License:* Public domain *Contributors:* Image created by JRM *Original artist:* JRM
- **File:Text_document_with_red_question_mark.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/a/a4/Text_document_with_red_question_mark.svg *License:* Public domain *Contributors:* Created by bdesham with Inkscape; based upon Text-x-generic.svg from the Tango project. *Original artist:* Benjamin D. Esham (bdesham)

12.3 Content license

- Creative Commons Attribution-Share Alike 3.0