P vs NP vs NPC

**I. Polynomial time: P**

**a. Generation vs verification**

**b. Decision vs.Optimization(search)**

**c. Reduction**

**II. Polynomial-time verification: NP**

**III. NP-completeness**

**\*\*\*\*\*\*\*\*\*\***

**I. Polynomial time: P**

Trying to define a complexity class P of all problems solvable in polynomial time.

We'll say that an algorithm runs in Polynomial Time if for some constant c, its running time is O(n^c), where n is the size of the input.

- Fortunately, for most reasonable encodings, whether or not an algorithm runs in Polynomial time is independent of the encoding. (i.e. the algorithm is mostly not pseudopolynomial)

- Unreasonable encoding: numbers in unary (ok to use, but think of it as a completely different problem, with potentially very different complexity)

- E.g., input is a number k, and running time is Theta(k)

If k is given in unary, n=k, so running time is Theta(n)

If k is given in binary, n = lg k, so running time is??? [Theta(2^n)]

**a. Generation vs Verification**

Verification of checking algorithm C for a problem P takes an input instance I and a solution certificate S and returns valid/invalid as the answer.

**b. Decision vs. Optimization**

Note that we’re still working to define a class of problems solvable in polynomial time.

Optimization Decision

Shortest path: given graph find shortest path is there a path of length at

and nodes (and integer k) most k

**Decision problem:** To convert an optimization problem to its decision version, we add an extra parameter k and ask: Is there a solution of size **at least/at most k**? The answer would be YES/NO.

As another example following is decision version of finding max element in a list: Is there a maximum element with value at least M in the given list?

Such reformulation of problem is not always possible eg: you can’t reformulate sorting as a decision problem. However all optimization problems can be recast as decision problems and makes it easier for us to classify problems according to their inherent complexities.

To be able to use decision version without ambiguity, we first need to show it’s okay to focus on decision version, for this we need to understand the idea of reduction.

**b. Reduction**

Problem A is poly-time REDUCIBLE to problem B if given a poly-time algo for B, we can use it to produce a poly-time algo for A (written: A <=\_p B).

Definition includes:

1. An algorithm to translate problem isntance of A to problem instance B in polyn +
2. Solving B in polyn +
3. Translating solution of B to solution of A in polyn

=> total poly time reduction

Problem A is poly-time EQUIVALENT to problem B (written: A =\_p B) if A <=\_p B and B <=\_p A.

[Read: reduction from vertex cover to Independent set and vice versa, reduction from vertex cover to set cover.]

Reductions are transitive (constant number of intermediate reductions, each polynomial time adds upto polynomial.

\*\*\*\*\*\*\*\*\*\*\*\*

* **Decision problems as defined above are poly-time equivalent to their corresponding**

**Optimization (search) problems.**

*Proof:*

Part I: Decision <=p Optimization [obvious if can solve search in polynomial time, we can

solve decision too using same algorithm- find the optimal solution in polynomial

time and thus it’s size. For the decision version just return YES if solution size is

at most k, No otherwise.]

Part II: Optimization <=p Decision???

[First use binary search on k to find optimal value of k (solving decision version

O(lg(k)) times for which the answer is Yes. If the answer is No, then return No

solution for optimization version. If answer is Yes, (we have found the optimal

value of k) then, simply run verification algorithm for each of C(n,k) possible

solutions, return the solution as soon as it is verified.]

\*\*\*\*\*\*\*\*\*\*

Now we’re ready to define the **complexity class P:**

**P:** Decision problems solvable in polynomial time.

E.g., shortest path is in P, so is MST. Note that there are problems such as sorting or towers of Hanoi etc that although are not optimization problems are still verifiable in polynomial time.

\*\*\*\*\*\*\*\*\*\*

**II. Polynomial-time verification: NP**

(the problems for which verification is polynomial time in size of solution)

We also say that problems in NP are solvable in Non-deterministic Polynomial(NP) time.

A common misconception is NP means non polynomial- this is not true- there may be a polynomial algo, we simply aren’t able to find one. So NP is abbreviation for non deterministic polynomial.

Guess(non determinism) + verify in polyn time

What does this mean? Two ways of thinking of it-

**Version 1: Verifier based definition:**

Interested in algorithms that verify solutions

E.g., Given a tour (a path through) of the cities, verify that its length is at most k.

NP: The class of problems that may/may not be solvable in polynomial time but are at least verfiable in polynomial time. YES instances have short proofs that the answer is yes. "short proof" means that proof is of polynomial size and can be checked by a deterministic algorithm (by a deterministic Turing machine) also called a verifier in polynomial time. These proofs are often called "witnesses" or "certificates". Verifier takes as input I,W: problem instance, proof and returns Yes/No in polynomial time.

E.g., for 3-coloring, the witness that answer is YES is a coloring.

For TSP, the witness that there is a tour of length <= k is a tour.

On the other hand, there may be no short proof that the answer is NO if it is a NO-instance. We may have to check all witnesses(exponentially many) to see none of them evaluate to YES=> long proof

Thus NP is like: "I don't know how to find it, but I'll know it when I see it"

It's pretty clear that P is contained in NP. (Since P is not only verifiable but also solvable in polyn time)

Note that: Verifier can return No even if the problem instance is Yes instance, this will be when the witness is incorrect.

**Version 2: Algorithm based definition:**

A non-deterministic algorithm A is like an oracle- a probabilistic algorithm, but we only require a solution certificate x such that

if x is a YES instance, then Pr(A(x) = YES) > 0 i.e. If x is a yes instance then there exists a certificate that will verify to YES. There is non-zero liklihood of guessing this witness.

if x is a NO instance, then Pr(A(x) = YES) = 0. i.e. there does not exist any certificate that will verify as No, probability of finding such a certificate is zero.

Example: Vertex Cover: Given a graph G, is there a vertex cover of size at most k? The non-deterministic algo will be just guessing a subset of size at most k and then verifying if this subset covers all edges-- if so, output YES; if not, output NO". So, vertex cover is in NP.

An algorithm that solves a problem in nondeterministic polynomial time can run in polynomial time or exponential time depending on the choices it makes during execution. (An equivalent deterministic algorithm may chart all exponentially many possible paths that non deterministic algo could take and hence would be exponential)

--> Biggest open question in complexity theory is P=NP? Are they really different? If they are the same, that would mean that any problem with the property that you could \*verify\* a solution quickly also has a fast way of finding the solution. That seems to good to be true also years of research has not resulted in any optimism in that direction, therefore most people believe P != NP.

Unfortunately though, we haven’t been able to prove otherwise as well. Since it's very hard to prove that a fast algorithm for something does NOT exist. So, whether or not P=NP still an open problem.

\*\*\*\*\*\*\*\*\*\*

**III. NP-Completeness**

Easy to define complexity classes.

Challenge is to define interesting ones.

What makes a class interesting:

- robustness (under reasonable machine models, encodings, etc)

- complete problems

NP has complete problems: if one is in P then all of NP is in P

a. Definition

Problem Q is NP-complete means:

(1) it's in NP, and

(2) any other problem Q' in NP is poly-time reducible to Q.

Thus, to show Q is NP-complete, we reduce some other NP-complete problem Q' to it.

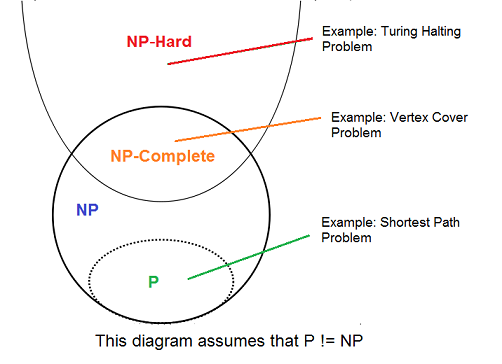
mapping Q' -----> Q or we write Q’ <=p Q

instance x' of Q' -----> instance x of Q (poly time)

x' is YES instance of Q' iff x is YES instance of Q.

NP-complete problems are in a sense the hardest problems in NP, since if you could solve Q in polynomial time, you could solve \*any\* problem in NP in polynomial time.

If problem satisfies (2) then it's called NP-hard.



\*\*\*\*\*\*\*\*\*\*

Our first NPC problem is Boolean satisfiability- A problem based on a boolean expression, asking if there’s an input assignment of variables so that the expression can evaluate to True.

1. This problem is in NP- given an assignment(proof), we can construct a deterministic turing machine that reads given boolean expression on it’s tape, evaluates it in polynomial number of steps and halts with answer Yes or No depending on whether or not the formula is satisfiable or not.[construction not in course]

2. All problems can be reduced to boolean Sat is proven by cook-levin theorem.

So, this is our first NP-complete problem. It's as least as hard as any problem in NP since in a sense any other computable problem is a special case of it.

\*\*\*\*\*\*\*\*\*\*

Advanced Reading [Not in course]

1. <https://pdfs.semanticscholar.org/7913/4aa1e23cb25fb5dd2d63af8342d6b7e792aa.pdf> (Turing machine and algorithms)
2. *Cook Levin’s theorem:*

[*https://dl.acm.org/citation.cfm?coll=GUIDE&dl=GUIDE&id=805047*](https://dl.acm.org/citation.cfm?coll=GUIDE&dl=GUIDE&id=805047)

1. Richard Karp’s landmark paper: (includes a simple reduction from boolean SAT to 3 SAT)

[*Karp, Richard M.*](https://en.wikipedia.org/wiki/Richard_Karp) *(1972). "Reducibility Among Combinatorial Problems". In Raymond E. Miller; James W. Thatcher.* [*Complexity of Computer Computations*](http://www.cs.berkeley.edu/~luca/cs172/karp.pdf) *(PDF). New York: Plenum. pp. 85–103.* [*ISBN*](https://en.wikipedia.org/wiki/International_Standard_Book_Number)[*0-306-30707-3*](https://en.wikipedia.org/wiki/Special:BookSources/0-306-30707-3)*.*

1. Garey and Johnson presented more than 300 NP-complete problems in their book *Computers and Intractability: A Guide to the Theory of NP-Completeness, (1979), W. H. Freeman.*