5 NP Completeness[1]

(Garey-Johnson slide)

We first give a small overview, and then in the remainder of the chapter, we will give exact definitions.

- P - class of problems that can be solved efficiently (i.e., in polynomial time)
- NP - class of problems that can solved in polynomial time by a non-deterministic Turing machine (nicer definition to follow)
- NP-complete problems - maximally difficult problems in NP: every other problem can be transformed into these in polynomial time
- NP-hard problems - like NP-complete problems, but not necessarily in NP

It is customary to depict these classes of problems in the following way (Fig. 5.1).

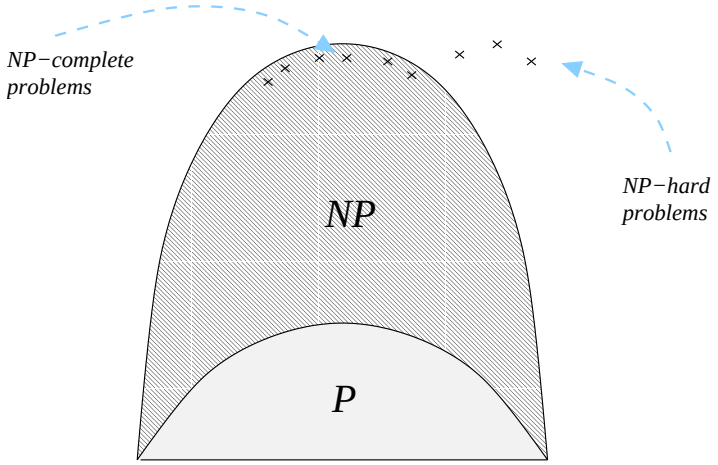

Figure 5.1: The classes of P and NP.

The big open question of theoretical computer science, and arguably of all of computer science is the following:

$$\text{Is } P = NP?$$

[1]Chapter 5 of Lecture notes by Zsuzsanna Lipták, zsuzsa@cebitec.uni-bielefeld.de, for "Selected Topics in Algorithmic Bioinformatics," Winter 2008/09, Bielefeld University - preliminary version of December 17, 2008

In other words, can every problem in NP be solved efficiently? Looking at Fig. 5.1, is the shaded part, the set NP \ P, empty? If you answer this question, you will instantly become the single most famous person in computer science, and you will also win a very large sum of money. The general consensus is that the answer is negative, i.e.,

Most people believe: $P \neq NP$.

Why do we believe this? Because there is a very interesting class of problems, the NP-complete problems, which are in some sense maximally difficult in NP. They have the following property:

Theorem 5.1. *Let X be an NP-complete problem. If we find a polynomial time algorithm for X, then* $P = NP$.

The reason is that every problem in NP can be "translated" into these problems (in polynomial time), so if we could solve these efficiently, then we could solve all other problems in NP efficiently, implying $P = NP$.

Many of the most interesting and widely applicable problems are NP-complete. But in spite of the huge amount of effort that has gone into trying to find efficient algorithms for these problems, over many decades by a large number of scientists, none have so far been found. This doesn't *prove* that there are none (we may just be too stupid), but it makes it appear more likely that there may be an inherent reason for this consistent failure: namely, that $P \neq NP$ and therefore no efficient algorithms exist for these problems.

A simple corollary from Theorem 5.1 is the following:

Theorem 5.2. *Let X be an NP-complete problem. Then there is no polynomial time algorithm for X, unless* $P = NP$.

The practical upshot of all this is: If we have established that a problem is NP-complete, then it is *very unlikely* that we will ever find an efficient algorithm solving it.

And what are NP-hard problems? They are just like NP-complete problems, except that we do not require that they be in NP themselves. They have the same "unlikely-to-be-efficiently-solvable" property:

Theorem 5.3. *Let X be an NP-hard problem. Then there is no polynomial time algorithm for X, unless* $P = NP$.

In the rest of the chapter, we will make these ideas precise.

5.1 Polynomial time transformations/reductions

So what do we mean by "translating" one problem into another?

(Little Prince example)

We will now define polynomial time reduction/transformation[2]. The idea is that $X \leq_p Y$ means: Any instance of X can be tranformed into an instance of Y in polynomial time, such that the new instance gives YES if and only if the old instance gives YES. The $\leq$ is supposed to imply that X is no more difficult than Y, and the p in the subscript refers to 'polynomial time.' More formally:

Definition 5.4 (polytime reducible)**.** We say that problem X is polynomial time reducible to problem Y, $X \leq_p Y$, if there is a polynomial p, and for every instance I of X, we can construct an instance J of Y in time $p(|I|)$ such that:

$$I \text{ is a YES-instance of } X \quad \Leftrightarrow \quad J \text{ is a YES-instance of } Y.$$

One can think of $X \leq_p Y$ as meaning: X is "not harder" than Y (with respect to polynomial time). Note that we are still paying for the reduction/transformation: it takes polynomial time!

Remark 1. By $|I|$, the size of I, we mean the actual number of bits it takes to encode I in binary encoding. For most practical purposes, though, it is enough to consider simpler parameters: $|V| + |E|$ for graphs $G = (V, E)$, number of entries for matrices, length $|s|$ for strings s, $\log_2 N$ for positive integers N, etc. This is o.k. because they only differ from the length of the binary encoding by a (usually constant) factor, which is not important in most cases.

We can use polynomial time transformations to find polynomial time algorithms for new problems:

Lemma 5.5. *If $X \leq_p Y$ and Y is polynomial time solvable, then so is X.*

The contraposition of this fact is what is used for NP-completeness proofs:

Lemma 5.6. *If $X \leq_p Y$ and X cannot be solved in polynomial time, then neither can Y.*

We will now look at an example.

5.2 A polynomial time transformation

We will show that CLIQUE $\leq_p$ COMPATIBILITY. We first need a formal definition of the two decision problems.

> CLIQUE (DECISION):
> Given a graph $G = (V, E)$, and an integer $k \leq |V|$, does G have a clique of size at least k?
>
> COMPATIBILITY (DECISION):
> Given a character state matrix M of dimension $n \times m$, and an integer $k \leq m$, is there a subset of the characters of size at least k which are pairwise compatible?

[2]The difference between 'reduction' and 'transformation' is subtle and is due to the fact that they have been introduced by different authors (Cook, 1971) and (Karp, 1972) - but of no importance for our purposes. We will slightly abuse notation $\leq_p$ and use the terms interchangeably.

Now given an instance of CLIQUE, we want to construct an instance of COMPATIBILITY. We will equate the vertices with characters, and will want to fill the matrix in such a way that two columns are compatible if and only if the corresponding vertices are connected by an edge.

The transformation:

Given an instance $G = (V, E)$ and $k \leq |V|$ of CLIQUE, we define a compatiblity matrix: The set of characters (the columns) equal V, and for each pair of vertices $u \neq v$ where $\{u, v\}$ is *not* in E, we define three objects $a_{u,v}, b_{u,v}, c_{u,v}$. Now we construct a forbidden submatrix for columns u, v and rows $a_{u,v}, b_{u,v}, c_{u,v}$, specifically: We set $M(a_{u,v}, u) = 1, M(a_{u,v}, v) = 0, M(b_{u,v}, u) = 0, M(b_{u,v}, v) = 1, M(c_{u,v}, u) = 1$, and $M(c_{u,v}, v) = 1$. All other entries of M are 0.[4]

Claim: In order to show that this is a valid transformation, we have to prove:

1. The answer is YES for the original instance (the graph has a clique of size at least k) if and only if the answer is YES for the new instance (there is a subset of compatible characters of size at least k).

2. The transformation can be done in polynomial time.

Proof. 1. It suffices to show: $\{u, v\} \in E$ if and only if c_u, c_v are compatible. Clearly, if $\{u, v\} \notin E$, then by construction, c_u, c_v are not compatible, since we have created a forbidden submatrix in the corresponding columns. Now assume that $\{u, v\} \in E$. We will show that for any row i, if $M_{i,u} = 1$, then $M_{i,v} = 0$, and, symmetrically, if $M_{i,v} = 1$, then $M_{i,u} = 0$. This implies that O_u and O_v are disjoint and thus compatible. So let i be a row s.t. $M_{i,u} = 1$. Then, $i = a_{u,w}$ or $i = b_{u,w}$ or $i = c_{u,w}$ for some $w \neq v$. Thus, $M_{i,v} = 0$.

2. For the transformation, we have to create 3 objects for each pair of vertices which are not connected by an edge, i.e., at most $3\binom{|V|}{2}$ many. The number of columns is $|V|$. Let $n = |V|$. So we have to fill in $\leq 3\binom{|V|}{2} \cdot |V| = O(n^3)$ many entries in the matrix, which is clearly polynomial in the input size $|V|+|E|$: $n \leq |V|+|E| \leq n+\frac{n^2}{2}$, since $0 \leq |E| \leq \binom{n}{2} \leq \frac{n^2}{2}$. □

5.3 Decision versus optimization problems

NP-complete problems are decision problems. The corresponding optimization problems are called NP-hard: They are not in NP (by definition, NP contains only decision problems), but they have the same property that they are unlikely to have an efficient algorithm (Theorem 5.3).

[4]The transformation works just as well if we define these 3 objects for *all* $u \neq v$. Then the forbidden submatrix is only created for those $u \neq v$ where $\{u, v\} \notin E$. For $\{u, v\} \in E$, *all* entries in rows $a_{u,v}, b_{u,v}, c_{u,v}$ are 0. The only difference is that the matrix has more rows.

	optimization	decision
some examples	Small Parsimony (Find value of best labelling)	Does M allow a PP?
	Large Parsimony (Find best tree)	Is M an ultrametric?
...	...	...
	COMPATIBILITY-opt (What is the size of a largest compatible subset?)	COMPATIBILITY-dec (Is there a compatible subset of size k?)

In many cases, if we can solve the decision version of a problem efficiently, then we can also solve the optimization version efficiently: If there is a known upper bound which is polynomial in n for possible values of k, then we can do binary search on possible values of k. Such an upper bound is, for example, n for the size of a clique, resulting in $O(\log n)$ calls to the polytime algorithm for the decision problem.

Thus, we only consider decision problems from now on.

5.4 NP: Polynomial time checkable problems

The reason why the "P=NP"-question is so important is this: Many problems of interest for which there is no efficient algorithm known are in the class NP: These are problems which are efficiently *checkable* (or *certifiable*). What does this mean? Given an instance I of a problem X. Let's assume that someone claims I is a YES-instance. If X is in NP, then there is a *certificate* γ (also called *proof*) such that γ proves that I is a YES-instance, and, moreover, this can be verified in polynomial time.

Some examples

problem	instance	certificate
CLIQUE	$G = (V, E), k$	a clique of size k
COMPATIBILITY	char-state matrix M, k	a compatible character set of size k
Small Parsimony	phylogenetic tree, cost c	a labelling with cost $\leq c$
Large Parsimony	char-state matrix M, cost c	a labelled tree with cost $\leq c$
PP	char-state matrix M	a PP
ultrametric	dist. matrix M	a rooted tree
additive	dist. matrix M	an unrooted tree

We now need some definitions:

Definition 5.7 (polytime algorithm)**.** Let A be an algorithm solving decision problem X. A is polynomial time if there is a polynomial p such that, for every instance I of X, A terminates in at most $p(|I|)$ many steps on input I.

Let X be a decision problem.

Definition 5.8 (P)**.** $X \in P$ if there is a polynomial time algorithm A solving X.

Definition 5.9 (NP)**.** $X \in NP$ if

1. There is a polynomial time algorithm B that takes two arguments I and γ, and
2. There is a polynomial p s.t. for every instance I of X: I is a YES-instance of X if and only if there exists a certificate ("proof") γ_I s.t.

$$|\gamma_I| \leq p(|I|) \text{ and } B(I, \gamma_I) = \text{ YES}.$$

In particular, B is polynomial time in $|I|$, i.e., there is a polynomial q s.t. B terminates after at most $q(|I|)$ many steps on input (I, γ_I).

Now the "P=NP"-question becomes: Can every problem for which a potential solution can be *checked* in polynomial time also be *solved* in polynomial time?

5.5 NP-complete problems

NP-complete problems are those problems in NP to which all other NP-problems can be reduced, i.e., if X is an NP-complete problem and $Y \in NP$, then $Y \leq_p X$. We recall the theorem from the beginning of this chapter:

Theorem 5.1 *Let X be an NP-complete problem. If we find a polynomial time algorithm for X, then $P = NP$.*

Proof. (Sketch) Let Y be any problem in NP. Since X is NP-complete, Y can be reduced to X (transformed into an instance of X) in polynomial time. But we have just found an algorithm for X which solves it in polynomial time. Polynomial time (transforming Y into X) plus polynomial time (solving the transformed Y) equals polynomial time. Altogether we have a polynomial time algorithm for Y, so $Y \in P$. □

In order to show that a problem X is NP-complete, we have to do two things:

1. Show that X is in NP.
2. Find a known NP-complete problem Y and a polynomial transformation of Y to X (i.e., show that $Y \leq_p X$).

When we say that a problem is NP-complete we are always implicitly referring to the theorem from the beginning of the chapter:

Theorem 5.2 *Let X be an NP-complete problem. Then there is no polynomial time algorithm for X, unless $P = NP$.*

And, since we all believe that $P \neq NP$, what people *actually* mean is this:

General Belief *Let X be an NP-complete problem. Then there is no polynomial time algorithm for X, fullstop.*

5.5.1 Proving that COMPATIBILITY is NP-complete

We have to first prove that COMPATIBILITY is in NP.

Given an instance M of COMPATIBILITY (decision) of size $m \times n$ and a number $k \leq m$, what would be a certificate for this instance? A subset of the character set of size k. Let C be such a subset. Then, $|C| = k \leq m$ and is thus polynomial in the input size (which is mn), choose e.g. polynomial $p(x) = x$. Checking the certificate takes time $\binom{|C|}{2} \cdot n = O(m^2 n)$, which is polynomial in $mn + k$, using e.g. polynomial $q(x) = x^2$.

Secondly, we have already shown that CLIQUE can be polynomially reduced to COMPATIBILITY. But CLIQUE is known to be NP-complete (see, e.g. the book by Garey and Johnson [GJ79]).

5.6 What now?

Let's assume we have established that the problem we are trying to solve is NP-complete or NP-hard. So we know that we are extremely unlikely to find an efficient algorithm for solving it. And now?

What to do when faced with an NP-complete or NP-hard problem:

- Despair/give up.
- Run an *exhaustive search algorithm:* This is often feasible on small instances (e.g. up to $n = 10$).
- Verify that your instances are really general. Often, the general case is NP-complete, but some *special cases* are not:

  For example, CLIQUE on general graphs is NP-complete, but for CLIQUE on planar graphs (graphs that can be drawn on a piece of paper without crossing edges), there is a polynomial-time algorithm.

  For example, PP is NP-complete, but PP for binary characters is not (recall the $O(nm)$ time algorithm from Section 4.2).
- Devise *heuristics:* algorithms that work well in practice but without a guarantee. (Runtime heuristics will often but not always run reasonably fast; other heuristics often but not always give a good output. Both without guaranteeing to do so!)
- Devise a polynomial time *approximation algorithm:* Here we compromise on the quality of the output but we usually have information about how much (approximation ratio).
- Devise a *randomized algorithm:* a non-deterministic algorithm, using random bits. Either the correctness or the running time are only given with a certain (known) probability. Thus, no guarantee but may work well in practice.

- Devise a *pseudo-polynomial time algorithm:* The runtime is a polynomial in the largest number of the input, e.g. if the input is a number K (thus, the input size is $\log K$, the runtime may be $O(K)$. This may be acceptable as long as the input numbers are not too large. (See Chapter 9 for an example.)

Literature for this chapter

The books by Kleinberg and Tardos [KT06] and by Cormen *et al.* [CLR90] have very nice chapters on NP completeness. The Garey-Johnson [GJ79] contains a huge list of NP complete problems; even though it is from 1979, it contains many problems that one encounters. Extended lists can be found on the internet. The transformation of CLIQUE to COMPATIBILITY was taken from the book by Setubal and Meidanis [SM97].

Bibliography

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability (A Guide to Theory of NP-Completeness)*. Freeman, New York, 1979.

[KT06] John Kleinberg and Éva Tardos. *Algorithm Design*. Addison Wesley, 2006.

[SM97] João Setubal and João Meidanis. *Introduction to Computational Molecular Biology*. PWS Boston, 1997.