1 coNP and good characterizations

In these lecture notes we discuss a complexity class called coNP and its relationship to P and NP. This discussion will lead to an interesting notion of "good characterizations" for problems, which plays an important role in algorithm design.

1.1 coNP

One property of the definition of NP is that YES and NO instances are not treated symmetrically. This asymmetry motivates the definition of coNP.

Definition.

$$\text{coNP} = \left\{\overline{L} \mid L \in \text{NP}\right\} \tag{1}$$

Here, $\overline{L} = \{0,1\}^* \setminus L$ is the set-theoretic complement of a language $L \subseteq \{0,1\}^*$. The language $\overline{L}$ represents the same decision problem as L except with all answers flipped. (NO instances in L are YES instance in $\overline{L}$ and the other way around.)

Example. Recall that SAT is the problem of deciding if a Boolean formula φ is satisfiable. As a language,

$$\text{SAT} = \{x \in \{0,1\}^* \mid \varphi_x \text{ is satisfiable}\}. \tag{2}$$

(Here, φ_x denotes the Boolean formula encoded by the string x.) We know that SAT $\in$ NP. Therefore, the complementary problem satisfies $\overline{\text{SAT}} \in$ coNP, where

$$\overline{\text{SAT}} = \{x \in \{0,1\}^* \mid \varphi_x \text{ is not satisfiable}\}. \tag{3}$$

A Boolean formula φ is called *tautology* if every assignment satisfies φ. The following problem is also in coNP,

$$\text{TAUT} = \{x \in \{0,1\}^* \mid \varphi_x \text{ is a tautology}\}. \tag{4}$$

This problem is essentially the same as $\overline{\text{SAT}}$ because a formula is not satisfiable if and only if its negation is a tautology.

1.2 Polynomial-time refuters

Similar to NP's definition in terms of polynomial-time verifiers, coNP has a definition in terms of polynomial-time refuters.

Lemma. A language $L \subseteq \{0,1\}^*$ satisfies $L \in$ coNP if and only if there exists a polynomial-time Turing machine R and a polynomial function $p\colon \mathbb{N} \to \mathbb{N}$ such that for every $x \in \{0,1\}^*$,

$$x \notin L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)}.R(x,u) = 0\,. \tag{5}$$

We say that a Turing machine R as in the lemma above is a *polynomial-time refuter* for L. For $x \in \{0,1\}^*$, we refer to strings $u \in \{0,1\}^{p(|x|)}$ with $R(x,u) = 0$ as *refutations* of the question

"$x \in L$?". (This notion of refutation for coNP is analogous to the notion of proof/certificate for NP.)

1.3 Short proofs and refutations

With the above terminology we can succinctly describe NP, coNP, and their intersection NP ∩ coNP:

$L \in$ NP**:** every YES instance of L has a *polynomial-length proof* of "$x \in L$?".
$L \in$ coNP**:** every NO instance of L has a *polynomial-length refutation* of "$x \in L$?".
$L \in$ NP∩coNP**:** every instance either has a polynomial-length proof or a polynomial-length refutation of "$x \in L$?".

1.4 NP vs coNP

The classes NP and coNP are widely believed to be different, which means that there exist problems such that YES instances have short proofs but NO instances do not necessarily have short refutations.

Conjecture. NP ≠ coNP.

The following lemma shows that proving this conjecture would also prove that P ≠ NP.

Lemma. If NP ≠ coNP then P ≠ NP.

Proof. If a language $L \subseteq \{0,1\}^*$ has a polynomial-time algorithm, then we can also decide $\overline{L}$ in polynomial time. Therefore, P = NP also implies that P = coNP.

The following lemma shows that NP-complete problems cannot have polynomial-length refutations unless NP = coNP.

Lemma. If NP ≠ coNP, then SAT ∉ coNP.

1.5 Efficient algorithms vs good characterizations

We say that languages $L \in$ NP ∩ coNP have a *good characterization* (that is, L has both short proofs and short refutations). Note that P ⊆ NP ∩ coNP. Therefore, a good characterization is necessary for a problem to have a polynomial-time algorithm. It turns out that in many cases the ideas behind a good characterization for a problem play an important role for designing polynomial-time algorithms. We will see some examples below.

There are different beliefs about whether P = NP ∩ coNP (researchers working in cryptography and complexity tend to believe P ≠ NP ∩ coNP whereas researchers working in algorithms and combinatorial optimization tend to believe P = NP ∩ coNP).

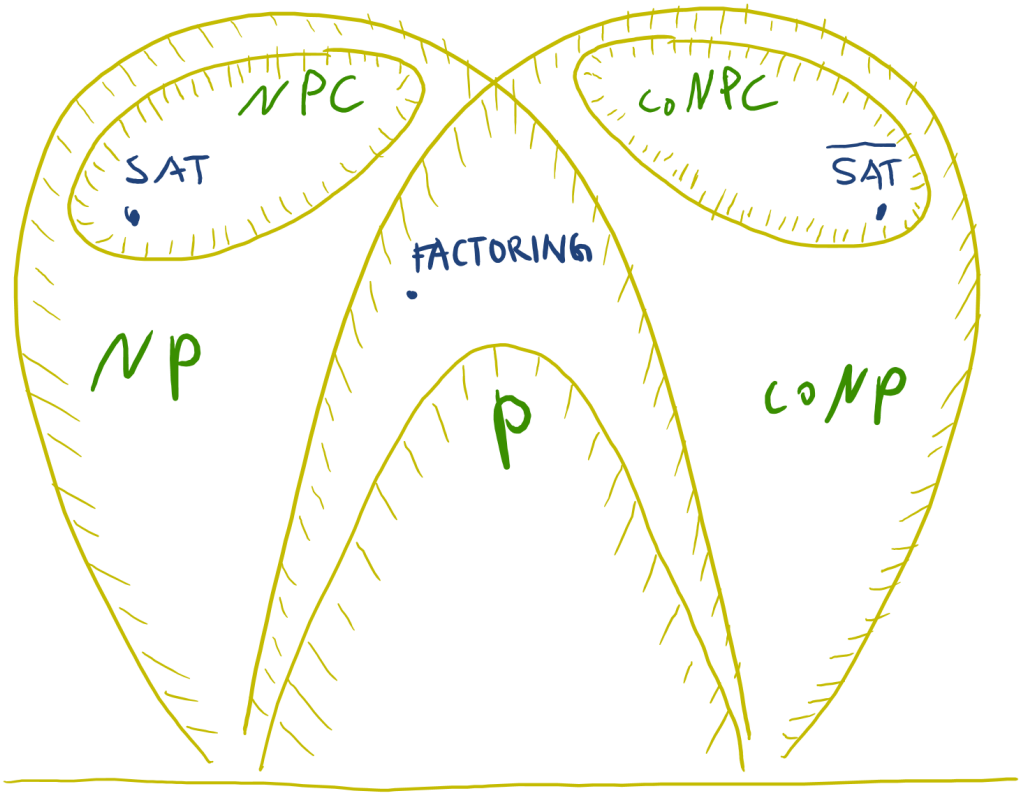
Question. P = NP ∩ coNP?

The following lemma shows that problems with good characterizations are unlikely to be NP-complete.

Lemma. If NP ≠ coNP, then no problem in NP ∩ coNP is NP-complete.

1.6 Conjectured landscape of NP vs coNP

The following sections give examples of problems with good characterizations. Some of these problems are also known to be in P (using more involved algorithms). Other problems, for instance FACTORING, are not known to be in P (and conjectured to be intractable by some researchers).



1.7 *Example:* Systems of linear equations

LINEQ. Given a system of linear equations, decide if the system is satisfiable. (More formally, we are given a matrix $A \in \mathbb{Q}^{m \times n}$ and a vector $b \in \mathbb{Q}^m$ —with all numbers in binary representation— and the goal is to decide if there exists a vector $x \in \mathbb{Q}^n$ such that $Ax = b$.)

We can solve this problem in polynomial-time by Gaussian elimination. It turns out that analyzing the running time of Gaussian elimination is quite subtle when taking into account bit complexity for arithmetic operations. [1]

We discuss how to show that LINEQ has a good characterization.

Theorem. LINEQ $\in$ NP $\cap$ coNP.

First, we show that the problem is in NP.

Lemma. LINEQ $\in$ NP.

Proof sketch. Given A, b, and x, checking $Ax = b$ takes time polynomial in the bit complexity of A, b, and x. (The fact that integer multiplication and addition take polynomial-time also implies that matrix-vector multiplication takes polynomial-time.) It remains to show that every satisfiable system of linear equations $\{Ax = b\}$ has a solution x with bit complexity polynomial in the bit complexity of A and b. This fact can be shown by bounds on determinants. See homework 5, question 3 for more detailed hints.

In order to show that LINEQ $\in$ coNP, we will use the following linear algebra fact.

Linear algebra fact. For every system of linear equations $\{Ax = b\}$ with $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}^m$, either there exists $x \in \mathbb{Q}^n$ with $Ax = b$ or there exists $y \in \mathbb{Q}^m$ such that $y^\top A = 0$ and $y^\top b = 1$.

Note that $y^\top A = 0$ and $y^\top b = 1$ implies that the system $\{Ax = b\}$ is not satisfiable, because every x satisfies $y^\top(Ax - b) = (y^\top A)x - y^\top b = 1$, which means that $Ax - b \neq 0$. We give a geometric proof for this fact at the end of this section.

The above linear algebra fact implies the following reduction between LINEQ and its complement.

Lemma. $\overline{\text{LINEQ}} \leq_p$ LINEQ.

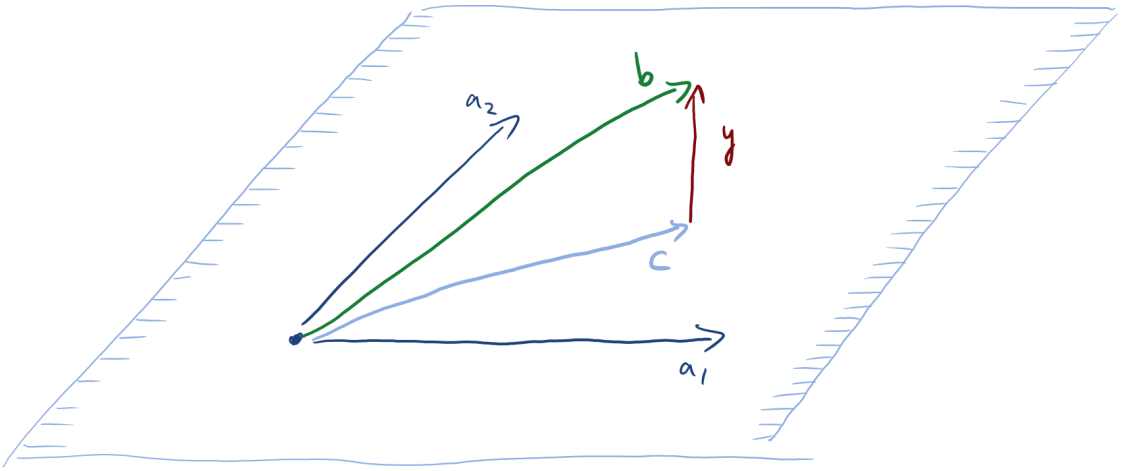
Proof. Let $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}^m$. By the above linear algebra fact, the linear system $\{Ax = b\}$ is not satisfiable if and only if the linear system $\{y^\top A = 0, y^\top b = 1\}$ is satisfiable. In matrix form, the latter system is $\{By = c\}$, where

$$B = \begin{pmatrix} A^\top \\ b^\top \end{pmatrix} \quad \text{and} \quad c = \begin{pmatrix} 0^n \\ 1 \end{pmatrix}. \tag{6}$$

It follows that the function mapping (A, x) to (B, c) as above is a polynomial-time Karp reduction from $\overline{\text{LINEQ}}$ to LINEQ

Proof of theorem. The previous lemmas—LINEQ $\in$ NP and $\overline{\text{LINEQ}} \leq_p$ LINEQ—together imply $\overline{\text{LINEQ}} \in$ NP (recall homework 3 question 2), which means that LINEQ $\in$ NP $\cap$ coNP.

Proof of linear algebra fact. Let $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}^m$. We are to show that the system $\{Ax = b\}$ has a solution if and only if the system $\{y^\top A = 0, y^\top b = 1\}$ does not have a solution. Let $U = \{Ax \mid x \in \mathbb{R}^n\}$ be the linear subspace of $\mathbb{R}^m$ spanned by the columns of A. We decompose b as the sum $b = c + y$ such that $c \in U$ and y is orthogonal to U. (Geometrically c is the point in U closest to b in Euclidean distance.) Since y is orthogonal to U, it satisfies $y^\top A = 0$ (which means geometrically that y is orthogonal to the columns of A).



By the construction of y, the system $\{Ax = b\}$ is satisfiable if and only if $y = 0$. Since $b = y + c$, the vector y satisfies

$$\|y\|_2^2 = y^\top y = y^\top (b - c) = y^\top b. \tag{7}$$

It follows that $y \neq 0$ if and only if $y^\top b \neq 0$. Therefore, by scaling y appropriately, we obtain a solution for $\{y^\top A = 0, y^\top b = 1\}$ if and only if $\{Ax = b\}$ is not satisfiable.

1.8 *Example:* Maximum flow and minimum cut

In the following, a *flow network* is a directed graph with dedicated source and sink, and nonnegative integer capacities assigned to edges.

MAXFLOW. Given a flow network G and a number $v \in \mathbb{N}$, decide if there exists a flow of value larger than v in G.

The following problem turns out to be closely related (by the max-flow min-cut theorem below).

MINCUT. Given a flow network G and a number $v \in \mathbb{N}$, decide if there exists a source-sink cut of capacity at most v in G.

Both MAXFLOW and MINCUT are problems in P (using fairly sophisticated combinatorial algorithms). Below we will discuss how to show that these problems have good characterizations.

Theorem. MAXFLOW, MINCUT $\in$ NP $\cap$ coNP.

Recall that the Ford–Fulkerson algorithm shows the following interesting-trivial fact.

Fact. Every flow network (with integer capacities) has a flow of maximum value that the flow values along all edges are integers.

This fact allows us to conclude that MAXFLOW $\in$ NP: Given a network G, a number v, and a potential flow f, we can verify in polynomial time if f is a valid flow in G of value larger than v. Furthermore, if there exists a flow of value larger than v, then there is also one with bit complexity bounded by the bit complexity of G. (Here, we use the fact above.) It is also straightforward to show that MINCUT is in NP.

Lemma. Both MAXFLOW and MINCUT are decision problems in NP.

Also recall that the analysis of the Ford–Fulkerson algorithm showed the following connection between MAXFLOW and MINCUT.

Maximum-flow minimum-cut theorem. For every flow network, the maximum value of a flow is equal to the minimum capacity of a source-sink cut.

Together with the previus lemma, the max-flow min-cut theorem allows us to show that MAXFLOW and MINCUT are also in coNP.

Lemma. $\overline{\text{MAXFLOW}} \leq_p \text{MINCUT}$ and $\overline{\text{MINCUT}} \leq_p \text{MINCUT}$.

Proof. In the problem $\overline{\text{MAXFLOW}}$, we are given a network G and a number v and we are to decide if the maximum flow in G is at most v. By the max-flow min-cut theorem, this question is equivalent to the question whether G has a source-sink cut of capacity at most v. It follows that $\overline{\text{MAXFLOW}} \leq_p \text{MINCUT}$. The argument for $\overline{\text{MINCUT}} \leq_p \text{MINCUT}$ is similar. (Note that we actually showed that $\overline{\text{MAXFLOW}}$ and MINCUT are the same problem in the sense $\overline{\text{MAXFLOW}} = \text{MINCUT}$.)

1.9 *Example:* Primality

Recall that a number is prime if it is greater than 1 and has no positive divisors other than 1 and itself. Here we consider the problem of testing if a given number is prime.

PRIMES. Given a number $p \in \mathbb{N}$ encoded in binary, decide if p is prime.

For this problem a good characterization has been known since 1976 (Pratt).

Theorem. PRIMES $\in$ P $\cap$ coNP.

Also polynomial-time *randomized* algorithms—by Miller–Rabin and Solovay–Strassen—have been known for this problem since around the same time. It has been an open problem to remove the use of randomness in these tests of primality. In 2002, Agrawal, Kayal, and Saxena showed that this problem indeed has a deterministic polynomial-time algorithm.

Theorem. PRIMES $\in$ P

1.10 *Example:* Integer factoring

The following problem plays a central role in cryptography (especially public-key encryption). An efficient algorithm for this problem could be used to break many cryptographic protocols used today, in particular RSA.

FACTORING. Given numbers $x, a, b \in \mathbb{N}$ encoded in binary, decide if there exists a prime factor p in the interval $[a, b]$ such that p divides x.

In order to give evidence for the security of current cryptographic protocols, it would be great to be able show that FACTORING is NP-complete. (Such a proof would not imply that the protocols are actually secure but it would still boost our confidence in their security.) However, it turns out that FACTORING has a good characterization, which means that it is not NP-complete unless NP is equal to coNP.

Theorem. FACTORING $\in$ NP $\cap$ coNP.

Proof. The problem is in NP because the prime factor p is a polynomial-length certificate for YES instances: Using the polynomial-time algorithm for PRIMES, we can verify in polynomial-time that p is a prime. We can also verify $p \in [a, b]$ and that p divides x.

The problem is also in coNP because a prime factorization of x is a polynomial-length certificate for NO instances: Given numbers $p_1, \ldots, p_k$, we can check that $p_1, \ldots, p_k$ is a prime factorization of x (again using the polynomial-time algorithm for PRIMES) and that none of the primes $p_1, \ldots, p_k$ are in the interval $[a, b]$. Here, we also use the fact that the prime factorization of a number is unique.

Footnotes

1. It is straightforward to show that the number of arithmetic operations performed by the algorithm is polynomial. What's more difficult to show is that these operations are only applied to numbers with polynomial bit complexity.