Mathematics and Computation Glossary

Carlos María Rodríguez

May 8, 2021

1 Basics

Def. 1.1. *$\boldsymbol{I}$ denotes the set of all binary sequences of all lengths. $\boldsymbol{I_n}$ denotes the sequence of all binary sequences in $\boldsymbol{I}$, this is, $\boldsymbol{I_n} = \{0,1\}^n$.*

Def. 1.2. *(The class $\boldsymbol{\mathcal{P}}$).*
A function $f : \boldsymbol{I} \to \boldsymbol{I}$ is in the class $\mathcal{P}$ if there is an algorithm computing f and positive constants, A, c, such that $\forall n \in \mathbb{N}, \forall x \in I_n$, the algorithm computes $f(x)$ in at most An^c steps.

Def. 1.3. *(The class $\boldsymbol{\mathcal{NP}}$).*
The set $\mathcal{C} \in \boldsymbol{I}$ is in the class $\mathcal{NP}$ if there is a function $V_C \in \mathcal{P}$ and a constant $k \in \mathbb{R}$ such that:

- *If $x \in \mathcal{C}$, then $\exists y \in \mathbb{R}$ with $|y| \leq k|x|^k$ and $V_C(x,y) = 1$.*
- *If $x \notin \mathcal{P}$, then $\forall y$ we have $V_C(x,y) = 0$.*

The function V_C is called the *verification algorithm*, and the sequence y for which $V_C(x,y) = 1$ is called the *witness*.

Def. 1.4. *(The class $\boldsymbol{co\mathcal{NP}}$).*
A set $\mathcal{C} \in \boldsymbol{I}$ is in the class $co\mathcal{NP}$ iff its complement $\bar{\mathcal{C}} = \boldsymbol{I} \setminus \mathcal{C}$ is in $\mathcal{P}$.

Def. 1.5. *(Efficient reductions).*
Let $C, D \subset \boldsymbol{I}$ be two classification problems. $f : \boldsymbol{I} \to \boldsymbol{I}$ is an efficient reduction from C to D if $f \in \mathcal{P}$ and $\forall x \in \boldsymbol{I}$ we have $x \in C \iff x \in D$.
We write $C \leq D$ if such a reduction exists.

Def. 1.6. *(Hardness and completeness).*
A problem D is called $\mathcal{C}$-hard if $\forall C \in \mathcal{C}$, we have $C \leq D$. If we further have that $D \in \mathcal{C}$, then D is called $\mathcal{C}$-complete.

Def. 1.7. *(The SAT problem).*
Given a logical expression over Boolean variables (can take values in $\{0,1\}$ with connectives $\wedge, \vee, \neg$), is it satisfiable? This is, is there a boolean assignment of the variables trough which the expression evalutates to 1? The set of all such expressions is denoted by SAT.

Theorem 1.1. *SAT is $\mathcal{NP}$-complete.*

Def. 1.8. *(The 2DIO problem).*
Given a Diophantine equations of the form $Ax^2 + By + C = 0; A, B, C \in \mathbb{Z}$, is it solvable with positive integers?

Theorem 1.2. *2DIO is $\mathcal{NP}$-complete.*

Def. 1.9. *(The 3COL problem).*
Given a planar map, can you color it using only 3 different colors?

Theorem 1.3. *3COL is $\mathcal{NP}$-complete.*

Def. 1.10. *(The $subset - sum$ problem).*
Given a sequence $a_1, a_2, ..., a_n \in \mathbb{Z}$ and b, is there a subset J such that $\sum_{i \in J} a_i = b$?

Theorem 1.4. *$Subset - sum$ is $\mathcal{NP}$-complete.*

2 Problems and classes related to $\mathcal{NP}$

Let us look at a few problem classes which do not fall into $\mathcal{NP}$ but which arise naturally.

Def. 2.1. *(Optimization problems).*
Fix a $\mathcal{NP}$ problem and a cost function over solutions (witnesses). Given an input, find the best *solution for it (minimizes cost function).*

Def. 2.2. *(Quantified problems).*
How does a problem from $\mathcal{NP}$ behave when parametrizing some of its variables? For example, which Boolean expressions have have valid solutions when considering every value possible for some given variables? The complexity class of all such problems is called the Polynomial Hierarchy, *denoted $\mathcal{PH}$.*

Def. 2.3. *(Counting problems).*
For a given $\mathcal{NP}$ problem and an input, find the number *of solutions (witnesses). The class of these problems is denoted by $\#\mathcal{P}$.*

Def. 2.4. *(Strategic problems).*
Given a (complete information, 2-player) game, find an optimal strategt for a given player. The basic class for these problems is denoted by $\mathcal{PSPACE}$, solvable using a polynomial amount of space.

Def. 2.5. *(Total $\mathcal{NP}$ functions).*
These are search problems seeking to find objects that are guaranteed to exist and are certified by small witnesses. These kind of problems can be divided into multiple complexity classes, lying between $\mathcal{P}$ and $\mathcal{NP}$. As examples:

- *Class $\mathcal{PLS}$, for polynomial local search.*
- *Class $\mathcal{PPAD}$, in which a problem consists in finding a fixed point for a given function.*
- *Computing a Nash equilibrium for a given 2-player game.*

Theorem 2.1. *If $\mathcal{P} \neq \mathcal{NP}$, there are infinitely many levels of difficuly in $\mathcal{NP}$.*

Some examples of problems which we haven't been proven to be in $\mathcal{P}$ yet aren't $\mathcal{NP}$-complete are:

- Integer factoring.
- Knot triviality. Given a diagram describing a knot, is it the trivial knot?
- Graph isomorphism. Given two graphs, are they isomorphic

Def. 2.6. *(Constraint satisfaction problems, CSPs).*
Fixing arity k (the locality parameter), alphabet Σ (possible values for the variables), and a relation $R \subseteq \Sigma^k$ (defining the set of tuple values satifying the constrint). We denote by $CSP(k, \Sigma, R)$ the following computational problem. Given a collection of k-tuples from a set of n variables, is there an assignment of the variables from Σ^n that satisfies all constraints in R?

Theorem 2.2. *(Dichotomy theorem). Every CSP is either in $\mathcal{P}$ or is $\mathcal{NP}$-complete.*

Def. 2.7. *(Unique Games).*
Fix $\epsilon > 0$ and integer m. The problem $UG(\epsilon, m)$ is the following. The input is a system of linear equations in n variables $x_1, x_2, x_3, ..., x_n$ over $\mathbb{Z}_m$, with two *variables per equation. An algorithm must answer "yes" is there is an assignment satisfying a fractoin $1 - \epsilon$ of the equations, and answer "no" if no assignment satisfies more than a fraction ϵ of them, any answer is acceptable if neither of these is the case.*

This is a specific case of CSP.

Conjecture 2.1. *(UGC). For every $\epsilon > 0$ there exists m such that $UG(\epsilon, m)$ is $\mathcal{NP}$-hard.*

Theorem 2.3. *Assume UGC. Then for every CSP, there is a constant δ such that approximating it to within approximation ratio δ is in $\mathcal{P}$, but approximating it to any better ratio $\delta + \epsilon$ is in $\mathcal{NP}$-hard for every $\epsilon > 0$.*

In some cases we may want to study how a given problem behaves depending on how its inputs are distributed, not only the worst-case scenarios. For example, we may be interested in how difficult a given problem is *on average*. One can generalize this notion considering *distributional* problems, given by (C, D); C is a classification problem, and D is a distribution on $\boldsymbol{I}$.

Def. 2.8. *(dist$\mathcal{P}$).*
We loosely define the distributional analog of $\mathcal{P}$, denoted by dist$\mathcal{P}$ as the set of problems having fast algorithms "on average".

Def. 2.9. *(One-way function).*
A function $f : \boldsymbol{I} \to \boldsymbol{I}$ is called one-way if $f \in \mathcal{P}$, but for every efficient algorithm A, its probability of computing any pre-image of f applied to a random input is small. Namely, for every (large enough) n,

$$Pr[f(A(f(x))) = f(x)] \leq \delta, \delta < 1$$

where the probability is taken over the uniform distribution of n-bit sequences x.

As an example, we may consider *modular exponentiation.* Let p be a prime, g a generator of $\mathbb{Z}_p^*$, and $ME_{p,g} : \{1, 2, ..., p-1\} \to \{1, 2, ..., p-1\}$, given by $ME_{p,g}(x) = g^{x-1} \bmod p$. Computing $ME_{p,g}$ is easy on every input, while it is believed that computing the inverse of $ME_{p,g}$ for a well-selected p is exponentially hard. We can combine these permutations into a single permutation ME:$\boldsymbol{I} \to \boldsymbol{I}$ and conjecture

Conjecture 2.2. *The modular exponentiation function ME is a one-way function.*

A key observation is that the existence of any one-way function would imply that dist$\mathcal{P} \neq$ dist$\mathcal{NP}$. In fact, it would prove that $\mathcal{NP} \cap co\mathcal{NP} \neq \mathcal{P}$.

Another important one-way function candidate is modular powering. Let p, q be primes, $N = pq$, and c invertible modulo $\phi(N) = (p-1)(q-1)$. Define $\mathrm{MP_{N,c}} : \mathbb{Z}_N \to \mathbb{Z}_N$ by $\mathrm{MP_{N,c}}(x) = x^c \bmod N$. This too is a permutation. Computing $\mathrm{MP_{N,c}}$ is easy, but the inverse isn't. Once again, we can construct a function MP: $\boldsymbol{I} \to \boldsymbol{I}$.

Conjecture 2.3. *The modular powering function MP is a one-way function.*

The difference between ME and MP is that MP has a *trap-door*: it becomes easy to invert if one has the factors of N. This allows anyone who has the secret key (the prime factors p, q) to restore the original message from a transformed one.

3 Lower bounds, Boolean circuits, and attacks on $\mathcal{P}$ vs. $\mathcal{NP}$

Def. 3.1. *(Boolean circuit).*
A Boolean circuit may be viewed as the hardware analog of an algorithm. Computation of a circuit on the Boolean inputs proceeds by applying a sequence of Boolean operations (gates) to produce an output. The operations are AND, OR, and NOT.

Def. 3.2. *(Circuit size).*
For a finite function g, denote by $S(g))$ the size of the smallest Boolean circuit computing g. More generally, for $f : \boldsymbol{I} \to \boldsymbol{I}$ with f_n the restriction of f to inputs of size n, we define $S(f)$ to be the mapping from n to $S(f_n)$.

Def. 3.3. *(The class $\mathcal{P}$/poly).*
$\mathcal{P}$/poly is the set of all functions computable by a family of polynomial-size circuits.

Theorem 3.1. *$\mathcal{P} \subseteq \mathcal{P}$/poly.*

This is because a given f_n function, one can simulate it's Turing machine, which runs in polynomial time T, with a circuit c_n whose size is $O((T(n))^2)$.

Conjecture 3.1. *$\mathcal{NP} \nsubseteq \mathcal{P}$/poly.*

Theorem 3.2. *For almost every function $f : \boldsymbol{I}_n \to \{0,1\}, S(f) \geq 2^{n/3}$.*

This is, most Boolean functions require exponential-size circuits. This can be seen by considering that for a number n of input bits, there are exactly 2^{2n} functions, while the number of different circuits of a given size s is about 2^{s2}. Each circuit computes a different function, so we must have $s > 2^{n/3}$ for most functions.
Sadly, this is a constructive proof, and we do not have any explicit function f with this property.

Conjecture 3.2. *$S(SAT) = 2^{\Omega(n)}$.*

This conjecture is muth stronger than $\mathcal{P}$ vs. $\mathcal{NP}$, as it would imply that SAT isn't in $\mathcal{P}$. But the fact is that *not a single nontrivial lower bound is known for a explicit function.* Formally,

Open Problem 3.1. *Find an explicit function $f : \boldsymbol{I}_n \to \boldsymbol{I}_n$ for which $S(f) \neq O(n)$.*

This means we are unable to provide any nontrivial lower bound, and we must turn to restricted problems to find relevant results.

We first focus on Boolean formulas, with the set of de Morgan connectors.

Def. 3.4. *(Formula size).*
For a finite function $g : \boldsymbol{I}_n \to \{0,1\}$, denote by $L(g)$ the size of the smallest Boolean formula computing g. For $f : \boldsymbol{I} \to \{0,1\}$, with f_n the restriction of f to inputs of size n, we define $L(f)$ to be the mapping of n to $L(f_n)$.

This notion is univeral, as every Boolean function can be computed by a Boolean formula. But, formulas are a worse computational model that circuits, and we can hope to prove better lower bounds. As an example, $S(\text{PAR}) = O(n)$, while $L(\text{PAR}) = O(n^2)$.

Theorem 3.3. *There is a Boolean function f with $S(f) = O(n)$ and $L(f) = \Omega(n^{3-o(1)})$.*

The actual gap is believed to be exponential:

Conjecture 3.3. *There is a Boolean function f with $S(f) = O(n)$ and $L(f) \neq n^{O(1)}$.*

Def. 3.5. *(CLIQUE problem).*
Given a graph on n vertices, is there a complete subgraph of size $\sqrt{n}$?

Theorem 3.4. *CLIQUE is $\mathcal{NP}$-complete.*

Def. 3.6. *(Perfect Matching problem).*
Given a graph, can you pair up the vertices so that every pair is connected by an edge?

Theorem 3.5. *Perfect matching is in $\mathcal{P}$*

Def. 3.7. *(Monotone function).*
A Boolean function is monotone if "increasing" the input (switching bits from 0 to 1) does not decrease the function value.

CLIQUE is one of such functions: adding edges cannot remove a complete subgraph. A similar reasoning can prove the same for Perfect matching.

Def. 3.8. *(Monotone circuit).*
A monotone circuit is one constructed only with the $\{\wedge, \vee\}$ gates.

It can be shown that any monotone function can be computed by a monotone circuit.

Def. 3.9. *(Monotone formula).*
A monotone formula is one constructed only with the $\{\wedge, \vee\}$ operators.

Theorem 3.6. *CLIQUE requires exponential-size monotone circuits.*

Theorem 3.7. *Perfect matching requires super-polynomial-size monotone circuits.*

Theorem 3.8. *Perfect matching requires exponential-size monotone formulas.*