National University of Singapore
School of Computing
CS3243

A Quick Guide to Computational Complexity

1 Decision Problems

The world is full of problems; some are harder than others. You may have previously seen problems with a polynomial time complexity in prior classes. In this class we will briefly explore problems that are *computationally intractable*.

Consider, for example, the following problem: we are given a social network graph; does this graph contain a group of $\geq k$ people that all know each other? More formally, we are given a graph $G = \langle V, E \rangle$ and a number k; does G contain a subset of vertices $C \subseteq V$ of size $\geq k$, such that for every two vertices $u, v \in C$, $\{u, v\} \in E$? This class of problems is known as CLIQUE, and is given by a graph $G = \langle V, E \rangle$, and a number k. Note that CLIQUE is a *decision problem* since every instance (i.e. a triple of the form $\langle V, E, k \rangle$) is either a `yes` instance — the graph $G = \langle V, E \rangle$ has a clique of size $\geq k$ — or a `no` instance — G does not have a clique of size $\geq k$.

More generally, a decision problem $\mathcal{P}$ comprises of a number of *instances*. An instance $I \in \mathcal{P}$ is either a `yes` instance or a `no` instance; we let $\mathcal{P}_{\texttt{yes}}$ be the set of `yes` instances in $\mathcal{P}$, and $\mathcal{P}_{\texttt{no}}$ be the set of `no` instances. We are interested in algorithms that can distinguish between `yes` and `no` instances in $\mathcal{P}$, i.e. given an instance $I \in \mathcal{P}$, decide whether $I \in \mathcal{P}_{\texttt{yes}}$ or in $\mathcal{P}_{\texttt{no}}$.

2 Complexity Classes

The *size* of an instance of the CLIQUE problem refers to the number of bits one would need in order to encode this problem in binary. For example, if the graph G has n vertices, we may consider encoding it using an $n \times n$ incidence matrix, requiring n^2 bits; the number k, encoded in binary, will need $\log k$ bits; thus the size of the instance is thus $\mathcal{O}(n^2 + \log k)$; since k has to be less than n (otherwise the CLIQUE instance is trivially a `no` instance), the total size of an instance of the CLIQUE problem is $\mathcal{O}(n^2)$. More generally, let $|I|$ be the number of bits needed to represent the instance I.

An algorithm $f : \mathcal{P} \to \{\texttt{yes}, \texttt{no}\}$ *decides* $\mathcal{P}$ if for every $I \in \mathcal{P}$, $f(I) = \texttt{yes}$ if and only if $I \in \mathcal{P}_{\texttt{yes}}$.

We say that the decision problem $\mathcal{P}$ is in `P` (the *polynomial complexity class*) if there exists an algorithm f that decides $\mathcal{P}$, and a constant k such that for every instance $I \in \mathcal{P}$, the time it takes to compute $f(I)$ is in $\mathcal{O}(|I|^k)$. In other words, there is some algorithm that can efficiently decide whether $I \in \mathcal{P}$ is a `yes` or `no` instance.

There is no known efficient algorithm that decides the CLIQUE problem. A "brute-force" approach would take an instance $\langle V, E, k\rangle \in$ CLIQUE and check whether any of the $\binom{|V|}{k}$ subsets of k vertices is a clique; however, there are $\Omega(n^k)$ such subsets, so this approach would potentially take time exponential in the number of bits needed to represent $\langle V, E, k\rangle$! However, the CLIQUE problem has an interesting property: given a candidate solution (i.e. a subset $C \subseteq V$ of $\geq k$ vertices), it is easy to verify in polynomial time whether it is a clique in G. In other words, there is some verification procedure we can run on C and $\langle V, E, k\rangle$ that outputs `yes` if $\langle V, E, k\rangle$ is a `yes` instance.
More generally, we say that a problem $\mathcal{P}$ can be verified in polynomial time if for every $I \in \mathcal{P}$ there exists some *witness* $w(I)$ such that $|w(I)|$ is polynomial in $|I|$, and $w(I)$ outputs `yes` on some poly-time verification procedure if $I \in \mathcal{P}_{\texttt{yes}}$. The class of problems that can be verified in polynomial time is called `NP`[1].
It is important to note that `no` witnesses are trickier. Just because we observe one set of vertices $C \subseteq V$ that is not a clique, does *not* mean that *every* set of $\geq k$ vertices is not a clique!

3 NP Completeness

Problems in `NP` are problems for which it is easy to check whether a candidate solution is correct in polynomial time; problems in `P` are ones for which there exists a procedure that finds a solution in polynomial time. It is easy to verify that `P` $\subseteq$ `NP`: for every instance $I \in \mathcal{P}$, the trace of the poly-time algorithm (with the `yes`/`no` bit appended to it) as your witness. The key question in computational complexity is whether:

$$\texttt{P} = \texttt{NP}$$

In other words, is checking solutions as difficult as actually coming up with them? There are some compelling reasons to believe that `P` $\neq$ `NP` (as well as compelling reasons to believe that `P` $=$ `NP`!); however, there is currently no definitive proof that this is the case.
Polynomial reductions between classes are a way of establishing a hierarchy of problem complexity. Given two decision problems $\mathcal{P}$ and $\mathcal{P}'$, we say that $\mathcal{P}$ reduces to $\mathcal{P}'$ if there exists a function $r : \mathcal{P} \to \mathcal{P}'$ (i.e. r takes as input some instance $I \in \mathcal{P}$, and outputs an instance $r(I) \in \mathcal{P}'$) such that for every instance $I \in \mathcal{P}$

1. $|r(I)|$ is polynomial in $|I|$.
2. the time that it takes to construct $r(I)$ is polynomial in $|I|$.
3. $r(I) \in \mathcal{P}'_{\texttt{yes}}$ if and only if $I \in \mathcal{P}_{\texttt{yes}}$.

[1] As an aside, the letters in `NP` stand for non-deterministic polynomial time. This echos the original definition of the class, which comprises of all decision problems that can be encoded on non-deterministic Turing machines that terminate in polynomial time. Amazingly, the (more intuitive) poly-time witness definition is equivalent.

In other words, a reduction takes an instance I and transforms it into an instance $r(I)$ that is of approximately the same size as I, and requires a reasonable time to compute. In addition, $r(I)$ is a `yes` instance if and only if I is.

Example 3.1. An instance of the *independent set* (INDSET) problem is given by an undirected, unweighted graph $G = \langle V, E\rangle$ and an integer k; it is a `yes` instance if and only if G contains an independent set of size $\geq k$. We say that a set of vertices $C \subseteq V$ is independent if for every $u, v \in C$, $\{u, v\} \notin E$.
Let us now describe a reduction from the INDSET problem to the CLIQUE problem: we map an instance $\langle V, E, k\rangle$ of the INDSET problem to an instance $\langle V, \bar{E}, k\rangle$ of the CLIQUE problem; V and k remain the same, but the set of edges $\bar{E}$ is the complement of E, i.e. $\{u, v\} \in \bar{E}$ if and only if $\{u, v\} \notin E$ (this is also unimaginatively referred to as the complement graph of $\langle V, E\rangle$). If $\langle V, E, k\rangle$ is a `yes` instance of INDSET, then there is some set $C \subseteq V$ of $\geq k$ vertices that have no common edge; under $\langle V, \bar{E}\rangle$, every two vertices in C will share an edge as $\bar{E}$ is the complement of E. Similarly, if $\langle V, \bar{E}, k\rangle$ is a `yes` instance of CLIQUE, then $\langle V, E, k\rangle$ is a `yes` instance of INDSET (the argument is symmetric).

Example 3.1 tells us that the INDSET is *at least as hard* as the CLIQUE problem: suppose that we have some efficient algorithm (call it A) that decides the CLIQUE problem; we can use the reduction r : INDSET $\rightarrow$ CLIQUE described in Example 3.1 to transform an instance $I \in$ INDSET to an instance $r(I) \in$ CLIQUE, and then compute $A(r(I))$ to decide whether $r(I)$ is a `yes` or `no` instance. Since $r(I)$ is a `yes` instance if and only if I is, the output of $A(r(I))$ is `yes` if and only if I is a `yes` instance. Since computing $r(I)$ takes polynomial time, $|r(I)|$ is polynomial in I, and A is a poly-time algorithm, the whole procedure takes polynomial time to compute!
We say that a decision problem $\mathcal{P}$ is `NP` hard if every problem in `NP` can be reduced to $\mathcal{P}$; we say that $\mathcal{P}$ is `NP` complete if it is `NP` hard and in addition, it is in `NP`. Many problems are `NP` complete; of particular note is the Boolean satisfiability (known as SAT) problem (the first problem proven to be `NP` complete): given a CNF formula ϕ over n Boolean variables $x_1, \ldots, x_n$, does there exist some true/false assignment to the variables (say $x_1 = \ell_1, \ldots, x_n = \ell_n$ for some $\ell_1, \ldots, \ell_n \in \{0, 1\}$) such that $\phi(\ell_1, \ldots, \ell_n)$ is true? It is easy to verify whether a candidate solution (i.e. truth assignment) to $I \in$ SAT is correct, thus SAT is `NP` complete. Other problems have been subsequently proven to be `NP` complete (the CLIQUE and INDSET are just two examples of such problems).
If we can show that even one of these `NP` complete problems can be decided by a poly time algorithm, then we have resolved `P` vs. `NP` in the affirmative! If one `NP` complete problem is poly-time decidable, then *every* problem in `NP` is! Some `NP` complete problems seem deceptively easy, yet no one has found a poly-time algorithm to solve them. Alternatively, if one can show that at least one `NP` complete problem is not poly-time decidable, then this will establish that `P` $\neq$ `NP`. In general, proving that a problem is `NP` complete makes a compelling case that it is highly non-trivial to solve.