Algorithmic Lower Bounds - Computational Complexity Crash Course

Matt Kovacs-Deak

December 11, 2021

Let Σ be some finite non-empty set, called the *alphabet*. Finite (ordered) sequences of elements Σ will be called *words* over the alphabet. Let Σ^* denote the collection of all words over Σ. Then for any subset $L \subseteq \Sigma^*$ we'll say that L is a formal language over the alphabet Σ. Now fix some alphabet Σ.

Definition 1. *The complexity class* $\mathbf{P}$ *is defined as the collection of all languages* L *for which,*

(i) There exists a Turing machine M *that runs in polynomial time on all of its inputs, and*

(ii) Given any $x \in \Sigma^*$ M *accepts if* $x \in L$*, and rejects otherwise.*

A few notes are in order:

- The running time of M is quantified in terms of the number of steps M takes as a function of the input size $|x|$, ie. the length of the input word. Notice that since Turing machines operate with discrete steps, this makes sense.
- To demonstrate that a language L belongs to $\mathbf{P}$, one can produce an algorithm $\mathcal{A}$ that decides L. A crucial point here is that the *Church-Turing Thesis* asserts that the intuitive notion of algorithms is the same as the class of algorithms captured by Turing machines.
- Conversely to disprove that $L \in \mathbf{P}$ one needs a *model of computation*. One such model is given by Turing machines. See Sipser's standard text for a formal definition.
- For the rest of this note, the terms *language* and *(decision) problem* will be used interchangably.

Analogously to $\mathbf{P}$ one can define the class $\mathbf{FP}$ of so called *function problems* that can be solved in polynomial time. Informally speaking a function problem seeks to efficiently evaluate some given function on a set of inputs.

One main goal in the study of complexity theory is to relate computational problems in terms of their *hardness* to other problems. This is done through so called *reductions*:

Definition 2. *Let* L_1 *and* L_2 *be some languages. Then we say that* L_1 *reduces polynomially to* L_2 *if there exists some* $f \in \mathbf{FP}$ *such that, for all* $x \in \Sigma^*$*,*

$$x \in L_1 \Leftrightarrow f(x) \in L_2$$

Whenever the above statement holds, the notation $L_1 \leq_p L_2$ *is used to signify this relationship.*

The intuition that is captured by the above definition is that L_2 is "at least as hard as" L_1 is. It is in easy exercise to prove the following properties: (1) The relation $L_1 \leq_p L_2$ is transitive, (2) If $L_2 \in \mathbf{P}$ and $L_1 \leq_p L_2$ then $L_1 \in \mathbf{P}$. Note that the contrapositive of the second point can be used to disprove membership in $\mathbf{P}$.

The Boolean Satisfiability Problem (SATISFIABILITY) is the following decision problem: Given a Boolean formula $\phi(x_1, ...x_n)$ is there a truth assignment for the variables x_i under which ϕ evaluates to 1? Notice that an exhaustive search would solve SATISFIABILITY in 2^n time. On the other hand given any truth assignment $a : \{x_i\} \to \{0, 1\}$ one can easily verify whether or not $\phi(a(x_1), ..., a(x_n)) = 1$. This notion of easy verifiability of candidate solutions is captured by the complexity class $\mathbf{NP}$,

Definition 3. *Let L be some language. $L \in \mathbf{NP}$ if there exists $L' \in \mathbf{P}$ together with some polynomial $\mathbf{P}$, such that,*

$$L_1 = \{x : \exists y : |y| \leq p(|x|) \text{ and } x.y \in B\}.$$

The intuition captured by the above definition is the following: for any $x \in L_1$ there exists a *witness* y that is not too long such that y can be used to efficiently verify the membership $x \in L_1$. On the other hand when $x \notin L_1$ the above definition doesn't guarantee any witness. Notice also that $\mathbf{P} \subseteq \mathbf{NP}$ clearly. As an exercise one can show that,

Exercise 4. *Show that $\mathbf{NP}$ is closed under union and concatenation.*

Below we give a couple examples of problems that belong to $\mathbf{NP}$:

(i) Given a graph G together with some $k \in \mathbb{N}$, does G contain a clique of size k?

(ii) Given a graph G together with some $k \in \mathbb{N}$, does G have a vertex cover of size k?

(iii) Given a graph G does it have an Eulerian cycle (one which visits each edge precisely once)?

(iv) Given $n, a \in \mathbb{N}$ is there some $2 \leq b \leq a$ that divides n?

(v) Given $S_1, ..., S_m \subseteq [n]$ and $k \in [m]$ are there distinct $i_1, ..., i_k \in [m]$ such that $[n] = \cup_j S_{i_j}$?

(vi) Given a multiset of integers S and a target integer t is there some $S' \subseteq S$ such that $\sum S' = T$?

The above problems are typically denoted by CLIQUE, EULER CIRCUIT, VERTEX COVER, `FACT`, SET COVER, SUBSET SUM respectively. The reader can easily convince herself that for any solution to the above problems there is an easy-to-verify witness, hence each of the above problems belongs $\mathbf{NP}$. Additionally, EULER CIRCUIT $\in \mathbf{P}$, for an Eulerian cycle exists if and only if each vertex has even degree. However, some of the above problems are *not known* to belong to $\mathbf{P}$. To capture this class of problems we will define,

Definition 5. *Let L be some language. L is said to be $\mathbf{NP}$-hard if, for all $L' \in \mathbf{NP}$, $L' \leq_p L$. If in addition $L \in \mathbf{NP}$, then we will say that L is $\mathbf{NP}$-complete.*

Cook and Levin independently proved that,

Theorem 6. SATISFIABILITY *is $\mathbf{NP}$-complete.*

The above thorem allows showing $\mathbf{NP}$-hardness without dealing with the computational model (e.g. Turing machines) directly: notice that one can prove that L is $\mathbf{NP}$-hard by producing a reduction from some L' which is already known to be $\mathbf{NP}$-hard. For example, by reducing from SATISFIABILITY one can show that,

Claim 7. *Each of the following problems are* **NP***-complete:* Clique, Vertex Cover, Set Cover, Subset Sum.

As noted earlier the inclusion $\mathbf{P} \subseteq \mathbf{NP}$ is easy to see. However, some problems (such as `FACT` from our list above) are not known to be in $\mathbf{P}$, while they clearly belong to $\mathbf{NP}$. Hence the question about the reverse containment $\mathbf{P} \supseteq \mathbf{NP}$ arises naturally. The answer to this question is not known, although most theorists believe that the inclusion $\mathbf{P} \subset \mathbf{NP}$ is proper. This makes intuitive sense, for one would expect that *finding* is generally harder than *verifying* (a solution).

Additionally, the supposition $\mathbf{P} \neq \mathbf{NP}$ can be used to explain the lack of certain results. In particular, there are computational problems for which efficient approximation algorithms are known up to a certain approximation ratio c, however despite many attempts no approximation algorithms are known beyond c. In many such cases it has been proved that surpassing a certain approximation ratio c would imply $\mathbf{P} = \mathbf{NP}$. Therefore, if $\mathbf{P} \neq \mathbf{NP}$ was indeed true, that would explain why one cannot produce approximation algorithms beyond certain ratios. For more details, see eg. [1].

Fixed parameter tractability

Consider the following two problems:

$$\text{Clique} = \{(G, k) : \text{ G has a clique of size } k\}$$
$$\text{Clique}_k = \{G : G \text{ has a clique of size } k\}$$

As discussed earlier Clique is **NP**-hard. On the other hand it is easy to see that there is a $O(n^k)$ algorithm that solves Clique_k. That is if we "freeze" the value of the parameter k, then Clique_k seems more tractable. This behavior is called *fixed parameter tractability*, and it characterizes many problems other than Clique, indeed another example would be Vertex Cover.

The Exponential Time Hypothesis

Notice that while the assumption Satisfiability $\notin \mathbf{P}$ rules out polynomial time algorithms for Satisfiability, it might still be plausible that there exists some $n^{O(\log n)}$ time algorithm that solves Satisfiability. The *Exponential Time Hypothesis* captures the widely held belief that Satisfiability in fact requires exponential time.

Strong NP-hardness

We claimed that `Subset Sum` is **NP**-hard, this is true as long as the input is encoded in binary. When its input is encoded in unary Subset Sum belongs to $\mathbf{P}$, as one can show that a dynammic programming algorithm can solve it efficiently. Thus, the notion of **NP**-hardness depends on the encoding of the inputs, to signify this the following terminology is introduced,

Definition 8. *A problem is weakly* **NP***-hard if it is* **NP***-hard when its inputs are given in binary encoding. When a weakly* **NP***-hard problem remains* **NP***-hard even when the inputs are encoded in unary, we say that the problem is strongly* **NP***-hard.*

One can analogously define *strong and weak* **NP***-completeness.*

Complexity of Function Problems

So far we mostly focused on decision problems, and we defined the complexity classes **P**, **NP**, etc. accordingly. These notions, however, are not suited for the study of so-called *function problems*. While for decision problems we are looking for a binary answer for each input, in the case of a *function problem* the output is typically more complex. As an example, one might formulate the function problem of finding the size of a maximal clique in a given graph. For such problems we have the class **FP**, the class of polynomially computable functions. Just as in the above example of CLIQUE, many decision problems have corresponding search variants. Furthermore, the complexities of the two variants are often related as suggested by the following claim,

Claim 9. *Let* MAX CLIQUE *denote the function problem of returning the size of the largest clique in a given graph. Then the following are equivalent,*

(i) CLIQUE $\in \mathbf{P}$

(ii) MAX CLIQUE $\in \mathbf{FP}$

The polynomial hierarchy

Say that two Boolean formulas are equivalent if they use the same set of variables, and their truth tables are identical. Consider the following decision problem called *Minimum Formula* (`MINFML`): Given a Boolean ϕ is there a shorter formula ψ that is equivalent to ϕ? As an example note that the following two formulas are equivalent $(a \vee d) \wedge (b \vee d) \wedge (c \vee d) \equiv (a \wedge b \wedge c) \vee d$, while the first one is longer. We rewrite this problem as,

$$\texttt{MINFML} = \{\phi : (\forall \psi \text{ with } |\psi| < |\phi|) : \exists \boldsymbol{b} : \phi(\boldsymbol{b}) \neq \psi(\boldsymbol{b})\}$$

Recall the definition of **NP**, and compare it to the above formulation. It seems that `MINFML` might not belong to **NP**. Indeed, one can define further complexity classes by generalizing the definition of **NP** by the way introducing additional quantifiers:

Definition 10. *We define the complexity classes* $\Sigma_1, \Pi_1, \Sigma_2, \Pi_2$ *by:*

- *Let* Σ_1*, which is also called* **NP***, be the set of decision problems* A *for which there exists some* $B \in \mathbf{P}$ *together with some polynomial* p *such that,*

$$A = \{x : \exists y_1 : |y_1| \leq p(|x|) \text{ and } (x, y_1) \in B\}$$

- *Let* Π_1*, also called* **coNP***, be the set of decision problems* A *for which there exists some* $B \in \mathbf{P}$ *together with some polynomial* p *such that,*

$$A = \{x : (\forall y_1 \text{ with } |y_1| \leq p(|x|)) : \ (x, y_1) \in B\}$$

- *Let* Σ_2 *be the set of decision problems* A *for which there is some* $B \in \mathbf{P}$ *together with a polynomial* p *such that,*

$$A = \{x : (\exists y_1 \text{ with } |y_1| \leq p(|x|)) : \ (\forall y_2 \text{ with } |y_2| \leq p(|x|)) : \ (x, y_1, y_2) \in B\}$$

- *Let* Π_2 *be the set of decision problems* A *for which there is some* $B \in \mathbf{P}$ *together with a polynomial* p *such that,*

$$A = \{x : (\forall y_1 \text{ with } |y_1| \leq p(|x|)) : \ (\exists y_2 \text{ with } |y_2| \leq p(|x|)) : \ (x, y_1, y_2) \in B\}$$

One can analogously define Σ_i and Π_i for $i = 3, 4, ...$; as well as the classes Σ_i-complete and Π_i-complete. Then the following containments are known collectively as *the polynomial hierarchy* (**PH**),

Claim 11 (The Polynomial Hierarchy)**.** *The following containments hold,*

- $\Sigma_1 \subseteq \Sigma_2 \subseteq \Sigma_3 \subseteq ...$
- $\Pi_1 \subseteq \Pi_2 \subseteq \Pi_3 \subseteq ...$
- $\Sigma_i \subseteq \Pi_{i+1}$
- $\Pi_i \subseteq \Sigma_{i+1}$

It is a straightforward exercise to show that if $\Sigma_i = \Pi_i$ for some $i \geq 1$ then the polynomial hierarchy collapses in the sense that $\Sigma_j = \Pi_j = \Sigma_{j+1}$ for all $j \geq i$. Note, however, that the polynomial hierarchy is believed to be proper.

Now consider the following modification of SATISFIABILITY: given a Boolean formula $\phi(\boldsymbol{x}, \boldsymbol{y})$ defined on two sets of variables, is there some assignment $\boldsymbol{x}'$ such that $\phi(\boldsymbol{x}', \boldsymbol{y}') = 0$ for all assignments $\boldsymbol{y}'$? Call this problem Σ_2-`SAT`. It can be shown that just like SATISFIABILITY is Σ_1-complete, so is Σ_2-`SAT` Σ_2-complete. This kind of construction is applicable more generally: it can be shown that one can modify any **NP**-complete problem to form a Σ_2-complete problem.

Intermediary problems

We noted above that EULER CIRCUIT $\in$ **P**, whereas CLIQUE, SATISFIABILITY, SUBSET SUM are all **NP**-complete. Often problems in **NP** either belong to **P** or are **NP**-complete. There are counterexamples to this pattern, however:

- `FACT` is the problem of given two integers $(n, a) \in \mathbb{N}$ is there a factor b of n with $b \leq a$? Currently `FACT` is not known to belong to **P**, neither is it known to be **NP**-complete. While it might be that `FACT` $\in$ **P**, it is unlikely to be **NP**-complete, as in that case **NP** = **coNP** would follow. Below we'll sketch a proof of this implication.
- GRAPH ISOMORPHISM is the problem of given two graphs determining whether or not they are isomorphic. Similarly to the case of `FACT`, the GRAPH ISOMORPHISM problem is not known to belong to **P**, and it's not known to be **NP**-complete either. And indeed, it seems unlikely that GRAPH ISOMORPHISM is **NP**-complete, for Boppana *et al* proved that in that case the polynomial hierarchy would collapse to level 2, ie. $\Sigma_2 = \Pi_2$ would follow.

Next we'll outline the main steps of the proof of the claim that if `FACT` is **NP**-complete, then **NP** = **coNP**.

Claim 12. *If `FACT` is* **NP***-complete, then* **NP** = **coNP**.

Proof (sketch). Let PRIMALITY denote the problem of determining whether a number is a prime. The proof proceeds in three steps,

1. Notice that PRIMALITY $\in$ **NP**.
2. Prove by PRIMALITY and the Fundamental Theorem of Arithmetic that $\overline{\texttt{FACT}} \in$ **NP**.
3. Notice that `FACT` $\in$ **NP** $\cap$ **coNP** to conclude the statement claimed.

■

In fact it is known that intermediate problems do exist provided that $\mathbf{P} \neq \mathbf{NP}$:

Theorem 13 (Ladner [8])**.** *If* $\mathbf{P} \neq \mathbf{NP}$ *then there is some* $L \in \mathbf{NP}$ *such that* $L \notin \mathbf{P}$ *and* L *is not* **NP***-complete.*

#SAT: number of satisfying assignments

A generalization of the `3-SAT` problem is the `#3SAT` problem: given a 3-CNF formula $\phi(x_1, ..., x_n)$ what is the number of satisfying assignments? Notice that this is clearly a harder problem than `3-SAT`, although it is not a decision problem anymore. The following two results relate `#3SAT` to other problems and the polynomial hierarchy **PH**:

- The *permanent* of a square matrix A is the quantity $\text{perm}(A) = \sum_{\sigma \in S_n} \prod_i a_{i,\sigma(i)}$. Valiant [11] showed that if computing the permanent is in **FP** then so is `#3SAT`.
- Toda [10] showed that if $\mathcal{A} \in \mathbf{PH}$, then $\mathcal{A}$ is reducible to `#3SAT`. Therefore, if `#3SAT` $\in \mathbf{FP}$, then every set in the polynomial hierarchy is in **P**.

Exponential time, and space complexity

So far we have only discussed time complexities of various problems. One can also classify problems based on how much space is required to solve them. To formalize this we define the following notions,

Definition 14.

- *Let* **PSPACE** *denote the set of decision problems which can be solved by a Turing machine that only uses a polynomial amount of its tape.*
- *Analogously to the above, let* **EXPSPACE** *denote the set of decision problems that can be solved using exponential space.*
- *Let* **EXPTIME** *denote the set of decision problems that can be solved by a Turing machine in exponential time.*

It is a straighforward exercise to show the following containments: $\mathbf{NP} \subseteq \mathbf{EXPTIME}$, and $\mathbf{NP} \in \mathbf{PSPACE}$.

Decidability

The complexity class **R** is defined to be the class of languages (ie. decision problems) that are *decidable* by a Turing machine, that is languages for which are *recognized* by some Turing machine that halts on all inputs. These languages have historically been called *recursive*, hence the notation. An example of an undecidable problem is the Halting Problem:

$$A_{TM} = \{\langle M, w\rangle : M \text{ is a Turing machine and it accepts } w\}$$

Another example of an undecidable problem is the following: Given $p \in \mathbb{Z}[x_1, ...x_n]$ is there some $\boldsymbol{a} \in \mathbb{Z}^n$ with $p(\boldsymbol{a}) = 0$? The status of the analogous problem for the rationals $\mathbb{Q}$ is not known.

Separations of complexity classes

A major goal in the study of computational complexity is to determine how various complexity classes relate to each other. Perhaps the most famous such question is the one about $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$. Unfortunately, there are only a few pairs of classes for which a separation is known. The following relations are known, however,

Theorem 15.

(i) $\mathbf{P} \subset \mathbf{EXPTIME} \subset \mathbf{R}$

(ii) $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXPTIME} \subset \mathbf{R}$

(iii) $\mathbf{P} \subseteq \mathbf{PSPACE} \subset \mathbf{EXPSPACE} \subset \mathbf{R}$

(iv) $\mathbf{P} \subset \mathbf{EXPTIME} \subset \mathbf{R}$

(v) $\mathbf{P} \subseteq \mathbf{NP}$ *and* $\mathbf{NP} \subseteq \mathbf{NPSPACE}$

(vi) $\mathbf{P} \subseteq \mathbf{PSPACE}$ *and* $\mathbf{PSPACE} \subseteq \mathbf{EXPSPACE}$

Notice that by part (i) of the above theorem, at least one of the inclusions in each of parts (v) and (vi) are proper.

Exercises

Exercise 16. *Recall the definition of* `MINFML`*. Show that assuming* $\mathbf{P} = \mathbf{NP}$*,* `MINFML` *is contained in* $\mathbf{P}$*.*

Exercise 17. *Recall the definitions of* **coNP** *and* **NP***, and show that* $\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{coNP}$*.*

Further reading

- In this note the algorithms (Turing machines) we considered were all deterministic, ie. they were not allowed to rely on stochasticity. One can formulate complexity classes where the algorithms considered are allowed to use randomness to produce their answers. Typically this means that they are not required to produce correct answers with probability 1. For example, analogous to $\mathbf{P}$ is the class **BPP** (abbreviating *Bounded-Error Probabilistic Polynomial Time*). Some language L is in **BPP** if there exists a probabilistic Turing machine M that (1) runs in polynomial time on all inputs, (2) accepts with probability no less than $2/3$ on inputs $x \in L$, (3) rejects with probability no less than $2/3$ on inputs $x \notin L$. Notice that $\mathbf{P} \subseteq \mathbf{BPP}$ clearly. However, it is not known how **BPP** is related to **NP**: we do not know if one is a superset of the other, or neither is contained in the other one. For some necessary conditions, and implications of $\mathbf{P} = \mathbf{BPP}$ see e.g. [6, 5].

- The complexity class **RE** (*recursively enumerable*) is defined to be the class of languages L for which a Turing machine can produce a list of all instances $x \in L$. Notice that $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{RE}$ is easy to see. It can be shown that the last inclusion is proper $\mathbf{NP} \subsetneq \mathbf{RE}$.

- The complexity class **IP** (abbreviating *Interactive Polynomial time*) was defined to comprise of problems for which a short *interactive* proof can be given. Informally speaking an *interactive proof system* for some language L involves a prover P and a verifier V such that given

some input word x, the prover and verifier interact through a shared tape with the objective that P proves to V that $x \in L$. It can be shown that our earlier example `#SAT` $\in$ **IP**, furthermore it was shown in 1990 by Shamir [9] that **IP** = **PSPACE**.

- Generalizing the class **IP** is the complexity class of **MIP** (abbreviating *Multi Prover Interactive Proof*). **MIP** is essentially the same as **IP** with the exception that the verifier is allowed to interact with many provers, not just one as in the case of **IP**. While the verifier is allowed to interact with each prover, the provers cannot interact with each other. Denoting by **MIP**$[k]$ the class where there are k provers, it has been shown [3] that **MIP** $\coloneqq$ **MIP**$[k]$ = **MIP**$[2]$ for $k > 2$. Babai, Fortnow and Lund [2] showed in 1991 that **MIP** = **NEXPTIME** the class of languages decidable in exponential time using a *non-deterministic* Turing machine.

- The complexity class **MIP*** is defined analogously to **MIP** with the exception that in this case the provers are allowed to share *quantum entanglement*[1]. Very recently Ji, Natarajan, Vidick, Wright and Yuen showed [7] that **MIP*** = **RE**. For a more accessible introduction see, e.g. [12], [13].

References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, May 1998.

[2] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *computational complexity*, 1(1):3–40, 1991.

[3] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi-prover interactive proofs: How to remove intractability assumptions. In *Proc. 20th STOC*, pages 113–131, 01 1988.

[4] A. Ekert, P. M. Hayden, and H. Inamori. Course 10: Basic Concepts in Quantum Computation. In R. Kaiser, C. Westbrook, and F. David, editors, *Coherent Atomic Matter Waves*, volume 72, page 661, Jan. 2001.

[5] O. Goldreich. In a world of $p = bpp$. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:135, 09 2010.

[6] R. Impagliazzo and A. Wigderson. P = bpp if e requires exponential circuits: Derandomizing the XOR lemma. In F. T. Leighton and P. W. Shor, editors, *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing*, pages 220–229. ACM, 1997.

[7] Z. Ji, A. Natarajan, T. Vidick, J. Wright, and H. Yuen. MIP*=RE. *arXiv e-prints*, page arXiv:2001.04383, Jan. 2020.

[8] R. Ladner. On the structure of polynomial time reducibility. *J. ACM*, 22:155–171, 1975.

[9] A. Shamir. Ip=pspace (interactive proof=polynomial space). In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 11–15 vol.1, Oct 1990.

[10] S. Toda. Pp is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20:865–877, 1991.

[11] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[12] D. Wakeham. MIP*=RE.

[13] H. Yuen. Mip* wiki.

[1] See [4] or `https://en.wikipedia.org/wiki/Quantum_entanglement`