Notes on Notions of Intractability

Gerard Jensen

January 6, 2025

Notes

This notes come from the Data Structures and Algorithims (EDA) course from Informatic Engineering at FIB, UPC during the Fall 2024 semester.

1 Introduction

Definition 1.1. A *decisional problem* Π is a computational problem which for every possible input x it is determined if a certain property is satisfied by x.
A *non decisional problem* is a computational problem where for each input x, the output is not binary (0 or 1).

Example 1.1.1.

- $\Pi \equiv$ Is G a Hamiltonian graph
- $\Pi \equiv$ Is it possible to colour G with k colours

Proposition 1.1. It is often easy to transform non decisional problems to decisional problems
Let a non decisional problem $\Gamma \equiv$ Find the shortest Hamiltonian cycle in a weighted graph G
Let a decisional problem $\Pi \equiv$ Is there a Hamiltonian cycle in a weighted graph G of weight $\leq W$
Binary search can be used to solve Γ from Π by changing W, where the cost of solving Γ equals the cost of solving Π multiplied by $\mathcal{O}((\text{size of input}))$

Definition 1.2. An input set E of a decisional problem Π is a set of binary strings encoding correct inputs of the problem. $E \subseteq (0+1)^* = \{\lambda, 0, 1, 00, 01, 10, \dots\}$

Definition 1.3. A set P of positive instances of a decisional problem Π is the set

$$P = \{x \in E \mid x \text{ satisfies the property } \Pi\}$$

Note: this definition mixes the concept of decisional problem and property, by referring at Π as a property. This is intentional. In general, there is no difference between the decisional problem and the property. Furthermore, the set P will also be used to reference a decisional problem, as it is the set of all objects that hold the property Π.

Definition 1.4. A *computational class* $\mathcal{C}$ is a set of decisional problems (in the set form). That is, a set of sets of positive instances of decisional problems.

Lemma 1.1. There exists some decisional problem P that can not be solved by any finite computational algorithm in finite time.

Proof. We first start by showing that the set of all finite programs (also called, Turing machines) has cardinality $\aleph_0$ (same as $|\mathbb{N}|$).
This is easy to show. Since any finite program can be represented as a finite string of 1's and 0's, and there exists a bijection between this type of string and the natural numbers (for instance, the binary representation of the string), we get that there are countably infinite different finite programs.

From definitions 1.2 and 1.3 we know that for a decisional problem P

$$P \subseteq (0+1)^*$$

Or, in other words

$$P \in \mathcal{P}\left((0+1)^*\right)$$

And the cardinality of the set $\mathcal{P}\left((0+1)^*\right)$ is $\mathfrak{c}$, or, the cardinality of the reals $|\mathbb{R}|$. Because any Turing machine can only decide (or solve) one decisional problem, by Cantor's Theorem we know that there must exist a decisional problem that no Turing machine can solve. □

Definition 1.5. An alphabet $\mathbb{A}$ is a set whose elements are called letters/symbols.
A sequence $\boldsymbol{x}$ of size n is an n-tuple $\boldsymbol{x} = (a_1, \ldots, a_n) \in \mathbb{A}^n$
An input set E is a subset of some alphabet $\mathbb{A}$, for instance $\mathbb{A} = \{0, 1\}$ Giving the mentioned before expression $E \subseteq \{0, 1\}^*$ where $\mathbb{A}^*$ is the set of every possible sequence of finite size using the symbols of $\mathbb{A}$
The size of $\boldsymbol{x}$ is denoted by $|\boldsymbol{x}|$

Corollary 1.1. All subsets of $\mathbb{A}^*$ for an alphabet $\mathbb{A}$ are totally ordered sets. This means that for $s_1, s_2 \in \mathbb{A}^*$ there exists a binary relation $\leq$ which can be used to order the elements of $\mathbb{A}^*$, for example $s_1 \leq s_2$ or $s_2 \leq s_1$.

2 Decidability

Definition 2.1. Given a decisional problem P, a (deterministic) algorithm $\mathcal{A}$ can be thought as the aplication

$$\begin{aligned} \mathcal{A} : E &\longrightarrow \{0, 1\} \\ x &\longmapsto \begin{cases} 1 & : \ x \in P \\ 0 & : \ x \notin P \end{cases} \end{aligned}$$

Definition 2.2. A decisional problem P with input set E is decidable in time $f : \mathbb{N} \to \mathbb{N}$ iff there exists an algorithm $\mathcal{A}$ $\forall x \in E$ cost of $\mathcal{A}(x) \in \mathcal{O}(f(|x|))$

Definition 2.3. Given an application f, the class TIME(f) is the set of all decisional problems P which are decidable in time f

$$\text{TIME}(f) = \{P \mid P \text{ decidable in time } f\}$$

Definition 2.4. The class P is the class of all decisional problems decidable in polynomial time. Defined as

$$\text{P} = \bigcup_{k \geq 0} \text{TIME}\left(n^k\right)$$

Definition 2.5. The class EXP is the class of all decisional problems decidable in exponential time. Defined as

$$\text{EXP} = \bigcup_{k \geq 0} \text{TIME}\left(2^{[n^k]}\right)$$

Corollary 2.1.
$\text{TIME}(n!) \subsetneq \text{TIME}(n^n) \subsetneq \text{EXP}$
$c^{n^k} \subsetneq \text{EXP}$

Theorem 2.1. $\text{P} \subsetneq \text{EXP}$

3 Nondeterminism

A non deterministic decidable algorithm is an algorithm where multiple paths are considered at once. Let E be the input set of the decisional problem of such algorithm. Given the input $x \in E$, each fork is made by a CHOOSE(y) function, which takes $y \in \{0,1\}^*$, and returns a z in the same set, satisfying that $\lambda \leq z \leq y \leq x^*$. For this multiple paths, if there exists one that would make the algorithm return `True` then the algorithm evaluates to 1. It does to 0 otherwise. $x*$ is not necessarily x, as x lives in the E space. However, it is something related (whatever it is programmed). The core idea of non deterministic algorithms is that they verify if a given input is a solution. For example, a non deterministic algorithm to calculate if a given number is not prime would return true if in one of the paths, the CHOOSE(y) function returns a divisor of such number. The algorithm only verifies that the output of CHOOSE(y) is a divisor of x or not.

On the other hand, a deterministic decidable algorithm doesn't use such function and there are no forks, the graph of states is a single path.

Definition 3.1. Lat $\mathbb{A} = \{0,1\}$ be the binary alphabet. A decisional problem P with input set E is decidable in non deterministic polynomial time iff $\exists k \in \mathbb{N}$, exists a deterministic polynomial time algorithm[1], called verifier $V : E \times \mathbb{A}^* \to \{0,1\}$ such that $\forall x \in E$:

$$x \in P \Rightarrow \exists c \in \mathbb{A}^* \ |c| \leq |x|^k \ V(x,c) = 1$$

$$x \notin P \Rightarrow \forall c \in \mathbb{A}^* \text{ s.t. } |c| \leq |x|^k \ V(x,c) = 0$$

c is called a certificate/witness that proves the problem.

What this definition is saying is that, to show that a decisional problem P with input set E can be solved in nondeterministic polynomial time, all positive inputs $x \in P$ must have some certifier y satisfying $|y| \leq |x|^k$ for some fixed $k \in \mathbb{N}$. Then, the verifier V algorithm must check in polynomial time weather if $x \in P$ for all $x \in E$ provided the certifier y.

In some sense, the function CHOOSE(y) represents a part (or all) of the certifier c, but it is not included in the formal definition.

Definition 3.2. The class NP is the class of all decisional problems decidable in nondeterministic polynomial time.

Theorem 3.1. $\text{P} \subseteq \text{NP}$

Proof. Given a decidable problem P in deterministic polynomial time ($P \in \text{P}$), with algorithm $\mathcal{A}$ and input set E. A verifier function V can be constructed such that $V(x,y) = \mathcal{A}(x) \ \forall x, y \in E$. Then, all values y are *trivial* certifiers of each $x \in P$. Which means that $\text{P} \subseteq \text{NP}$. □

[1] Which means, $V(x,y)$ can be decidable in time $p(|x| + |y|)$ for some polynomial $p(n)$.

4 Reductions

A reduction is a concept involving 2 decisional problems and one function.

Definition 4.1. Let A, B be 2 decisional problems (note, nothing is specified if those problems live in P, NP or wherever class) whose input spaces are E and E'. Let f be an application $f : E \to E'$ which computes in polynomial time. Let $\leq^p$ be a binary relation between A and B, which, for instance $A \leq^p B$ reads as: A reduces to B in polynomial time.

$$A \leq^p B \iff \exists f \in \mathcal{O}(n^k) s.t.$$

$$x \in A \Leftrightarrow f(x) \in B$$

for some $k \in \mathbb{N}$, and for n be the size of the input element in E.

What this is saying is that, given 2 decisional problems A, B, $A \leq^p B$ means that it is possible to solve A via B. The way of doing so is to get every input of A, transform it to an input of B, via function f and apply that input to solve B. Then, that binary output is the same as for the original A input. An equivalent definition would be:
Let $\mathcal{A}, \mathcal{B}$ be algorithms for decisional problems A, B with input spaces E, E':

$$A \leq^p B \text{ via } f \iff \forall e \in E \; \mathcal{A}(e) = \mathcal{B}(f(e))$$

One can view reductions as saying $A \leq^p B$ means that A is not harder than B because you can solve A through B adding polynomial time.

Theorem 4.1.

- $\leq^p$ is a preorder (it holds $\forall A \; A \leq^p A$ and $\forall B, C \; A \leq^p B, B \leq^p C \Rightarrow A \leq^p C$)
- For $\mathcal{C} \in \{\mathrm{P}, \mathrm{NP}\}$, $\leq^p$ is closed downwards which means that $A \leq^p B, B \in \mathcal{C} \Rightarrow A \in \mathcal{C}$
  What this states is that, since A is not harder than B you can solve A at least at the complexity of B in this classes.

Definition 4.2. The class NP-hard is the class of all decisional problems that NP problems can be solved through them. That is:

$$\text{NP-hard} = \{A \mid \forall P \in \mathrm{NP} \; P \leq^p A\}$$

Or, in other words, A is NP-hard if NP is a subset (may or not be proper) of all problems that can be solved through[2] A. That is: all NP problems reduce to A in polynomial time.

Note that for $A \in$ NP-hard doesn't imply that $A \in$ NP. If this is also true, then A belongs to the NPC class.

Definition 4.3. The class NPC, or NP-complete is the class of all decisional problems that are both NP-hard problems and NP problems. That is:

$$P \in \mathrm{NPC} \Leftrightarrow \{A \mid A \leq^p P\} = \mathrm{NP}$$

Corollary 4.1. The following equivalencies hold:

- $X \in$ NPC

[2] in polynomial time

- $X \in \text{NP}, X \in \text{NP-hard}$
- $\{A | A \leq^p X\} = \text{NP}$

The proof of the later proposition is:

Proof.

$$X \in \text{NPC} \Rightarrow X \in \text{NP}, X \in \text{NP-hard}$$

Applying the definition of NP-hard:

$$\text{NP} \subseteq \{A | A \leq^p X\}$$

Since X is in NP, applying theorem 4.1 (closure under NP):

$$A \leq^p X \in \text{NP} \Rightarrow A \in \text{NP}$$

Thus:

$$\text{NP} = \{A | A \leq^p X\}$$

□

Corollary 4.2. To prove that X is NP-hard it is sufficient to find a reduction of the type

$$A \leq^p X, A \in \text{NP-hard}$$

Proof. A is NP-hard, so all NP problems reduce to A. Since $A \leq^p X$, by transitivity all NP problems reduce to X so X is NP-hard. □

To prove that X is in P or NP, the inverse reduction (from X to a P or NP problem) must be found. This works because of closure under P and NP.

5 SAT and other problems

The satisfiability problem is a well know NPC problem. The algorithm must decide if a prepositional formula is satisfiable or not.

Definition 5.1. A propositional formula ϕ is an expression involving parentheses, the constants `true` and `false`, Boolean variables $x_0, \ldots, x_n$ which might take values true or false, their negations $\bar{x}$ and the connectives *lor* (or, disjunction) and $\wedge$ (and, conjunction).

Definition 5.2. For a propositional formula ϕ to be satisfiable means that $\exists x = (x_0, \ldots, x_n) \in \{\texttt{true}, \texttt{false}\}^n$ such that $\phi(x) = \texttt{true}$. That means that depending of the boolean variables, ϕ can be a `true`.

Theorem 5.1 (Cook-Levin's Theorem (1971))**.** The SAT problem is NPC.

Proof. SAT $\in$ NP because for a specific set of values of the boolean variables of a propositional formula, it can be checked in polynomial time the overall result.

Let's now prove that SAT is NP-Hard. That means, all NP problems reduce to SAT. Let $B \in \text{NP}$. This means that $\forall x \in B$ there exists a witness y of polynomial size of $|x|$ and a polynomial time verifier $V(x, y)$, such that, when y is a witness for input x, $V(x, y) = 1$. Now, for a specific x, we can construct the algorithm $V_x(y) = V(x, y)$. This V_x runs on hardware, that is: logic gates, logic

1's and 0's. So there exists an intrinsic propositional formula, denoted F_{V_x} that solves V_x for any y. Since $y \in \{0,1\}^*$, y can be represented with 1's and 0's. In our case, each 1 and 0 will be the boolean variables of F_{V_x}. The construction and size of this F_{V_x} is polynomial, but it won't be proved. Let $g_V(x) = F_{V_x}$ be the reduction function that takes input x and transforms into an input for SAT, our F_{V_x}. If F_{V_x} is satisfiable it means that a set of variables evaluate the expression to `true`. Those variables represent a witness y. This means that $B \leq^p$ SAT via g_V. So SAT is NP-hard. □

5.1 Other problems

Name	**Class**	**Description**
Clique	NPC	G graph contains a complete subgraph of order k.
Hamiltonian	NPC	G graph is Hamiltonian.
Independent set	NPC	G graph contains k vertices which aren't connected one to another.
VC: Vertex Cover	NPC	G graph contains k vertices so that every edge is incident to at least one of them.
$k > 2$-Colourability	NPC	G graph can be coloured by k colors such that 2 connected vertices don't have the same colour.
2-Colourability	P	G graph can be coloured by 2 colors such that 2 connected vertices don't have the same colour. Same as, if graph is bipartite.
Bin-packing	NPC	Items with sizes $d_1 \dots d_n$ and k bins of capacity m. All items to be packed into the bins at the same time.
Tripartite-matching	NPC	3 disjoint sets C_1, C_2, C_3 with n elements each and a subset $S \subseteq C_1 \times C_2 \times C_3$. S contains n 3-tuples s.t. all elements of the 3 sets are present.
Dominating set	NPC	G graph with $W \subseteq V$ vertices, $k = \lvert W \rvert$ s.t. $\forall v \in V \setminus W \; \exists w \in W v \sim w$.