Foundations of Probabilistic Proofs (Fall 2022)
Note 1: Intro to IPs
Date: 2022.09.20

This note contains definitions, theorems, facts, *etc.* that are not fully explained in lectures due to limited time. If you think there are anything missing or any mistakes, please contact ziyi.guan@epfl.ch.

1 Complexity classes

We introduce IP in the lecture and compare it with several other complexity classes. Here we review the definition of these classes. For an overall picture of the relationships between different complexity classes, you can refer to `https://complexityzoo.net/Complexity_Zoo`.

Definition 1 (Turing Machines)**.** *A Turing machine is a mathematical model of computation describing an abstract machine that manipulates symbols on a strip of tape according to a table of rules. It can be used to implement any algorithms. In this note, we can think of Turing machines as Python programs.*

Definition 2 (P, Polynomial-Time)**.** *The class of decision problems solvable in polynomial time by a Turing machine.*

- *What is decision problems?*
  - *Any function that maps a string to a Boolean value is a decision problem. More formally, let Σ be an alphabet, any $f\colon \Sigma^* \to \{0,1\}$ is a decision problem over Σ.*
- *On the other hand, we often claim that a language $\mathcal{L}$ is in some complexity classes. What is a language?*
  - *Any (possibly infinite) subset $\mathcal{L} \subseteq \Sigma^*$ is called a language over the alphabet Σ.*
- *There is a one-to-one correspondence between languages and decision problems.*
  - *Let $f\colon \Sigma^* \to \{0,1\}$ be some decision problem. Define $\mathcal{L}$ to be subset of Σ^* that f maps to 1.*
  - *Let $\mathcal{L} \subseteq \Sigma^*$. Define $f\colon \Sigma^* \to \{0,1\}$ such that $f(w) = 1$ iff $w \in \mathcal{L}$.*
- *Can you give some examples of decision problems/languages that are in P?*
  - *Decision version of sorting: decide wether a given array is sorted.*
  - *Determine if two vertices in a graph are connected.*

Definition 3 (NP, Nondeterministic Polynomial-Time)**.** *Fix some alphabet Σ. We say that a language $\mathcal{L}$ can be decided in non-deterministic polynomial time if there exists a polynomial-time Turing machine V that always halts and takes in two strings as input s.t. for all $x \in \Sigma^*$:*

1. *if $x \in \mathcal{L}$, then there exists $u \in \Sigma^*$ with $|u| \leq |x|^k$ for some constant k s.t. $V(x,u)$ accepts;*
2. *if $x \notin \mathcal{L}$, then for all $u \in \Sigma^*$, $V(x,u)$ rejects.*

NP *is the set of all languages which can be decided in non-deterministic polynomial time.*

- NP *can also be defined using non-deterministic Turing machines.*
  - *A non-deterministic Turing machine is a model of computation whose governing rules specify more than one possible action when in some given situations. In other words, deterministic Turing machines output one result for one input, but non-deterministic Turing machines can have multiple different computation paths (think about a computation tree) that lead to possibly different outputs.*
  - NP *is the set of all languages solvable by polynomial-time non-deterministic Turing machines:*
    - *if $x \in \mathcal{L}$, at least one computation path accepts;*
    - *if $x \notin \mathcal{L}$, all computation paths reject.*
- NP*-hardness and* NP*-completeness.*
  - *Polynomial-time many-one reduction (Karp reduction): Let $\mathcal{A}$ and $\mathcal{B}$ be two languages. Suppose that there is a polynomial-time computable function (also called a polynomial-time transformation) $f\colon \Sigma^* \to \Sigma^*$ such that $x \in \mathcal{A}$ if and only if $f(x) \in \mathcal{B}$. Then we say that there is a polynomial-time many-one reduction (or a Karp reduction, named after Richard Karp) from $\mathcal{A}$ to $\mathcal{B}$, and denote it by $\mathcal{A} \leq_m^P \mathcal{B}$.*

    Example 1 (CLIQUE $\leq_m^P$ IS)**.** *We first define the two languages.*

    $$\text{CLIQUE} = \{\langle G, k\rangle \colon G = (V, E) \text{ is a graph}, k \in \mathbb{N}^+, G \text{ contains a } k\text{-clique}\}$$

    where a k-clique is a complete graph with k vertices.

    $$\text{IS} = \{\langle G, k\rangle \colon G = (V, E) \text{ is a graph}, k \in \mathbb{N}^+, G \text{ contains an independent set of size } k\}$$

    where an independent set is a subgraph with no edges between any vertices.
    Consider the following function

    ```
    def f(<G=(V,E),k>): {
        E'={{u,v}:{u,v} not in E}
        return <G'=(V,E'),k>
    }
    ```

    One can verify that $x \in$ CLIQUE iff $f(x) \in$ IS.
  - *$\mathcal{L}$ is $\mathcal{C}$-hard if for all languages $\mathcal{K} \in \mathcal{C}$, $\mathcal{K} \leq_m^P \mathcal{L}$.*
  - *$\mathcal{L}$ is $\mathcal{C}$-complete if $\mathcal{L}$ is $\mathcal{C}$-hard and $\mathcal{L} \in \mathcal{C}$.*
  - *Cook-Levin theorem:* SAT *is* NP*-complete.*
    - SAT*: given a Boolean formula, determine if there exists a satisfying assignment.*
  - *To show a language $\mathcal{L}$ is* NP*-hard, it is sufficient to show $\mathcal{K} \leq_m^P \mathcal{L}$ for some* NP*-hard language $\mathcal{K}$.*

Definition 4 (coNP, Complement of NP)**.** *A language $\mathcal{L}$ is in* coNP *if $\mathcal{L}^c = \Sigma^* \setminus \mathcal{L}$ is in* NP*. Alternatively, we could define* coNP *as the set of languages s.t. for any string in the language, there exists a polynomial-time non-deterministic Turing machine where all computation paths accept.*

Definition 5 (MA, Merlin-Arthur)**.** *Given a decision problem, Merlin, who has unbounded computational resources, sends Arthur a polynomial-size proof that the answer to the problem is 1. Arthur must verify the proof in probabilistic polynomial-time (randomized version of* P*), so that*

1. *If the answer is 1, then there exists a proof such that Arthur accepts with probability at least 2/3.*
2. *If the answer is 0, then for all proofs Arthur accepts with probability at most 1/3.*

Definition 6 (PSPACE, Polynomial-Space)**.** *Similar to Definition 2,* PSPACE *is the set of languages solvable by a Turing machine in polynomial-space.*

- *Can you show that* NP $\subseteq$ PSPACE*?*

2 Equivalence relations

In the lecture, we claim that graph isomorphism is an equivalence relation. Here, we formally define equivalence relations and verify that graph isomorphism satisfies the definition.

Definition 7 (Equivalence Relations)**.** *Given a relation $\mathcal{R} : (X, X) \to \{0, 1\}$, it is an equivalence relation iff it satisfies the following conditions:*

- *reflexive: for all $a \in X$, $\mathcal{R}(a, a) = 1$;*
- *symmetric: for all $a, b \in X$, $\mathcal{R}(a, b) = 1$ iff $\mathcal{R}(b, a) = 1$;*
- *transitive: for all $a, b, c \in X$, if $\mathcal{R}(a, b) = 1$ and $\mathcal{R}(b, c) = 1$ then $\mathcal{R}(a, c) = 1$.*

The equivalence class of an element $a \in X$ is defined as $\{x \in X : \mathcal{R}(a, x) = 1\}$. Note that by the three constraints above, any element in X can be in at most one equivalence class.

Now we are ready to verify that graph isomorphism is an equivalence relation. Let GI be a relation that takes in two graphs $(G_1 = (V_1, E_1), G_2 = (V_2, E_2))$, and $\mathsf{GI}(G_1, G_2) = 1$ iff there is some bijective function $f : V_1 \to V_2$ *s.t.* $\{u, v\} \in E_1$ iff $\{f(u), f(v)\} \in E_2$. We show that GI satisfies the conditions:

- reflexive: for any graph G, $\mathsf{GI}(G, G) = 1$ since we can set f to be the identity function;
- symmetric: assume $\mathsf{GI}(G_1, G_2) = 1$ for some arbitrary G_1, G_2, and let f be the corresponding bijective function. We know that f^{-1} exists because f is bijective. Then $\mathsf{GI}(G_2, G_1) = 1$ with f^{-1} as the underlying bijective function.
- transitive: assume $\mathsf{GI}(G_1, G_2) = 1$ and $\mathsf{GI}(G_2, G_3) = 1$ for some arbitrary G_1, G_2, G_3, and let f, g be the corresponding bijective functions. Then $\mathsf{GI}(G_1, G_3) = 1$ with $f \circ g$ as the underlying bijective function.

3 Sequential repetitions v.s. parallel repetitions

We give the definition of IP in the lecture, in which completeness and soundness errors are defined. Now, for simplicity, let's assume that we are given an interactive proof protocol with perfect completeness (i.e. $\epsilon_{\mathsf{c}} = 1$). Our goal here is to reduce the soundness error. The usual approach is by repetition, which means running several independent instances of the protocol. However, this can be done sequentially or in parallel. We define and analyze each of them in this section.

3.1 Sequential repetitions

In a sequential repetition, the $(i+1)$-st execution of the protocol is only started after finishing the i-th execution.

Definition 8. *Given an interactive proof system (P, V), the* k**-sequential repetitions** *of it is a proof system (P_k, V_k) in which P_k and V_k engages in k independent executions of (P, V), one after another. V_k accepts if and only if all k V's accept.*

We note that if the soundness error for the original interactive proof system is ϵ_{s}, then after k-sequential repetitions, the soundness error of (P_k, V_k) would be ϵ_{s}^k, which can be arbitrarily small for large k.

Sequential repetitions are easy to define and analyse. However, one caveat is that it increases the round complexity of the proof system multiplicatively by a factor of k. This issue can be avoided by parallel repetitions.

3.2 Parallel repetitions

Parallel repetitions mean that all protocols are run simultaneously. We define it formally in the following.

Definition 9. *A* k**-parallel repetitions** *of an interactive proof system, (P, V), is a proof system (P_k, V_k) in which the parties play in parallel k copies of (P, V). The interaction between P_k and V_k is summarized in the following table:*

Prover P_k	***Verifier*** V_k
Generate randomness $\omega_1, \ldots, \omega_k$	*Generate randomness* $r_1, \ldots, r_k$
	Send $\beta_{(1,1)}, \ldots, \beta_{(k,1)}$ *s.t.* $\beta_{(i,1)} = V(r_1)$
Send $\alpha_{(1,1)}, \ldots, \alpha_{(k,1)}$ *s.t.* $\alpha_{(i,1)} = P(\omega_1, \beta_{(i,1)})$	
$\cdots$	
	Send $\beta_{(1,j)}, \ldots, \beta_{(k,j)}$ *s.t.* $\beta_{(i,j)} = V(r_j, \alpha_{(1,j)}, \ldots, \alpha_{(i-1,j)})$
Send $\alpha_{(1,j)}, \ldots, \alpha_{(k,j)}$ *s.t.* $\alpha_{(i,j)} = P(\omega_j, \beta_{(i,1)}, \ldots, \beta_{(i,j)})$	
	Check V accepts all k copies

Note that the round complexity of the original proof system is preserved. Now we are ready to analyse the soundness error for the parallel repetition.

Theorem 1 (Parallel repetition theorem)**.** *Let V be an interactive machine, and V_k be an interactive machine obtained from V by playing k versions of V in parallel. Let*

$$p(x) \coloneqq \max_{P^*} \{\Pr[(P^*, V)](x) = 1\},$$
$$p_k(x) \coloneqq \max_{P^*} \{\Pr[(P^*, V_k)](x) = 1\}.$$

Then we have $p_k(x) = p(x)^k$.

We omit the formal proof here, we refer curious readers to read through Appendix C.1 of `https://www.wisdom.weizmann.ac.il/~oded/PDF/mcppp-v2.pdf` for the complete proof; however, we explain the main proof idea and intuitively how it differs from sequential repetitions.

- Parallel repetition is not as trivial as sequential repetition: intuitively, it is easier for any malicious prover P_k to cheat by colluding within one round.
- $p_k(x) \geq p(x)^k$:
  - Let $\tilde{P} \coloneqq \arg\max_{P^*} \{\Pr[(P^*, V)](x) = 1\}$.
  - $p(x) = \Pr\left[(\tilde{P}, V)\right](x) = 1$.
  - $p_k(x) \geq \left(\Pr\left[(\tilde{P}, V)\right](x) = 1\right)^k = p(x)^k$.
- $p_k(x) \geq p(x)^k$: Exploit the notion of the ***game tree*** of a proof system $(\mathcal{P}, \mathcal{V})$.
  - Nodes of the game tree denote the partial transcripts of the verifier $\mathcal{V}$:
    - Root: empty interaction at level 0.
    - Internal nodes: edges going out from even/odd level nodes correspond to the messages $\mathcal{V}/\mathcal{P}$ may send given the content (transcript) of the node.
    - Leaves: complete transcripts.
  - Value of the tree denote the probability of the the path being accepting:
    - Leaves: 1 if the transcript is accepting, 0 otherwise.
    - Nodes with children:
      - even level: weighted average of the values of its children, where the weights is the probability of the corresponding messages being sent.
      - odd level: maximum of the values of its children (prover always trying to maximize the accepting probability).
  - Consider the game trees T, T_k for the original proof system (P, V) and the k-repeated proof system (P_k, V_k), respectively.
  - There is a natural 1-1 mapping of nodes in T_k to sequences of k nodes in T. Then prove by induction that the value of each node in T_k is the product of the values of the corresponding k nodes in T.