Probabilistic Proof Systems

CS6230: Topics in Information Security
Lecture 1 (Aug 10, 2021)
Lecture and notes: Prashant Nalini Vasudevan

1 Introduction

Proofs are a fundamental part of mathematics. They allow us to prove the truth of complex statements starting from a handful of basic axioms. Classically, proofs take the form of a sequence of inferences or assertions, each of which are easily verifiable, and which together imply the theorem being proven.

In this course, we will study proof systems that depart from this traditional notion by allowing for interaction and randomness. Research on such proof systems has led to many important advancements in theoretical computer science over the past 30 years. We will study their definitions, their power and properties, and their connections to various parts of theoretical computer science.

Pre-requisites. Students will need to be comfortable with computational complexity theory, as most of our study will build on concepts from there. Knowledge of basic probability theory and algebra will be necessary to follow most of the constructions and proofs. Knowledge of cryptographic theory will help, but will not be assumed.

1.1 Course Information

- Instructor: Prashant Nalini Vasudevan (prashant@comp.nus.edu.sg)
- Time: Tue 10am - noon SGT
- Location: Online for now – meeting links on LumiNUS
- Office Hours: Wed 10am - noon SGT – meeting links and booking slots on LumiNUS
- Grading: 3-4 Problem Sets (60%), and final project (40%)
- References: Lecture notes and references on course website
- Website: `https://www.comp.nus.edu.sg/~prashant/teaching/CS6230/`

1.2 Grading

Problem Sets (60%, distributed over 3-4 sets)

- Will be posted on course website and LumiNUS
- Collaboration encouraged, but your submission must be written on your own
- Submit on LumiNUS, in the relevant folder in the Files section
- No late submissions accepted without my explicit permission
- First problem set will be uploaded tomorrow, is due next Monday (Aug 16)

Final Project (40%)

- Read and write a survey/report on a few related papers
- Work in groups of 1 or 2
- Details will be announced in a few weeks

2 Course Overview

Proofs are a fundamental part of mathematics. They allow us to prove the truth of complex statements starting from a handful of basic axioms. Classically, proofs take the form of a sequence of inferences or assertions, each of which are easily verifiable, and which together imply the theorem being proven. Take, for example, the following simple proof of a well-known theorem.

Theorem 2.1. $(a+b)^2 = a^2 + 2ab + b^2$

Proof.

$$\begin{aligned}(a+b)^2 &= (a+b)\cdot(a+b) && \textit{(by definition)}\\ &= a\cdot a + b\cdot a + a\cdot b + b\cdot b && \textit{(distributivity of addition over multiplication)}\\ &= a^2 + 2ab + b^2 && \textit{(commutativity of multiplication)}\end{aligned}$$

□

Each equality here can be verified to be true in a straightforward manner, and together they prove the theorem. More generally, a classical proof of a theorem is any string that, when read, *convinces* the reader (or "verifier") that the theorem is true. We will refer to this system of proving theorems as a "classical proof system". Any such proof system ideally has three desirable properties:

- **Completeness:** Any true statement has a valid proof.
- **Soundness:** No false statement has a valid proof.
- **Efficient Verifiability:** The validity of a proof can be verified efficiently.

In this course, we will study proof systems that have weaker versions of the above properties. We will be covering four broad topics and their interactions, both among themselves, and with many other areas of computer science.

Interactive proofs: These extend the classical notion of a proof by adding interaction. Here, there is a "prover" that *interacts* with a "verifier", and tries to convince it that a given statement is true. In the simplest formulation, the prover is allowed unbounded computational power, but the verifier is required to be computationally efficient.

Zero-knowledge proofs: These are interactive proofs in which, in the course of the interaction, the verifier learns nothing more than the truth of the statement being proven. Just the definition of this property lead to a paradigm of security definitions that is now pervasive in cryptography.

Arguments: Also known as computationally sound proofs, these are interactive proofs in which the prover is required to also be computationally efficient (but may have information that the verifier does not). These, specifically highly efficient zero-knowledge arguments, have a variety of applications, from authentication protocols to blockchains.

Probabilistically Checkable Proofs: These are static proofs that may be verified without reading the entire proof. These have deep connections with complexity theory, and are also instrumental in constructing efficient arguments.

3 Proof Systems for Languages

We will be concerned, for the most part, with proof systems for languages. Recall that a language is a set of strings from $\{0,1\}^*$, usually those with some clearly stated property. For example, SAT is the set of all strings that represent Boolean formulas that are satisfiable.

Here, in the context of a language $L \subseteq \{0,1\}^*$, given an "input" $x \in \{0,1\}^*$, we are interested in proofs of theorems of the form $x \in L$. Without loss of generality, these proofs may also be represented by strings $y \in \{0,1\}^*$. We will represent the verifier of the proof by an algorithm V, and rewrite the above properties as follows,

- **Completeness:** For any $x \in L$, there exists a y such that $V(x, y)$ accepts.
- **Soundness:** For any $x \notin L$ there is no y such that $V(x, y)$ accepts.
- **Efficient Verifiability:** The verifier V runs in time polynomial in $|x|$.

Note that above, the only choice we made that loses generality is interpreting "can be verified efficiently" as "V runs in polynomial time". This choice is in keeping with the treatment of polynomial running-time as efficiency in complexity theory.

The class of languages L for which there are proof systems that prove $x \in L$ and have the above properties is NP, or Non-deterministic Polynomial time.

Exercise 1. *Prove that the above definition of* NP *is equivalent to the definition based on Non-deterministic Turing Machines.*

4 Randomness in Proofs

The first relaxation we will make to classical proofs is to allow the verifier V to be *randomised*, rather than deterministic. Then, we will allow the completeness and soundness conditions to fail with some small probability over the randomness of the verifier. The class of languages that have proof systems of this form is called MA (for reasons that will become clear later.)

Definition 4.1. MA consists of all languages L for which there exists probabilistic polynomial time (PPT) algorithm V (the "verifier") and a polynomial p satisfying the following properties:

- **Completeness:** For every $x \in L$, there exists a $y \in \{0, 1\}^{p(|x|)}$ such that:
$$\Pr_V [V(x, y) \text{ accepts}] \geq 2/3$$
- **Soundness:** For every $x \notin L$, and every $y \in \{0, 1\}^{p(|x|)}$:
$$\Pr_V [V(x, y) \text{ accepts}] \leq 1/3$$

Exercise 2. *Prove that the class* MA *would be unchanged if the* $2/3$ *and* $1/3$ *in Definition 4.1 were replaced by* $(1 - \delta)$ *and* δ*, respectively, for any constant* $\delta < 1/2$*. How small can* δ *be made without changing the definition?*

4.1 The Power of Randomised Verification

As an example of how access to randomness can help with verification, consider the task of verifying matrix multiplication. That is, given matrices $A, B, C \in \mathbb{F}^{n\times n}$ for some finite field $\mathbb{F}$, how can one verify that $C = A \cdot B$?

The simplest way is to actually perform the multiplication $A \cdot B$, and check whether the result is equal to C. This can be done in using roughly $O(n^{\omega})$ field operations, where the best known bound on ω is that it is at most 2.37826 [AW21]. Can this be done faster?

Turns out it can, following an elegant algorithm due to Freivalds [Fre77]. This works as follows:

1. Sample a random vector $v \leftarrow \mathbb{F}^n$.
2. Compute $u \leftarrow Bv$, and $w \leftarrow Au$.
3. Compute $z \leftarrow Cv$.
4. Accept $w = z$, and reject otherwise.

Note that this procedure only requires $O(n^2)$ field operations. If it is indeed the case that $C = A \cdot B$, then for any v we have $z = Cv = (A \cdot B)v = A(Bv) = Au = w$. Thus, the verification procedure will always acept, is termed "perfectly complete".

On the other hand, suppose that $C \neq A \cdot B$. In this case, we would like the procedure to reject with high probability. This is guaranteed by the following lemma if $\mathbb{F}$ is large enough.

Lemma 4.1. *For matrices $C, D \in \mathbb{F}^n$ such that $C \neq D$, the probability that $Cv = Dv$ for a uniformly random vector $v \leftarrow \mathbb{F}^n$ is at most $1/|\mathbb{F}|$.*

Proof. If $C \neq D$, then there exists some $i \in [n]$ such that the i^{th} rows of C and D are different. Denote these rows by C_i and D_i, respectively. Denote by $(Cv)_i$ and $(Dv)_i$ the i^{th} coordinates of Cv and Dv. In order for Cv and Dv to equal, $(Cv)_i$ and $(Dv)_i$ have to be equal. We will prove that the probability of this happening is low.

If $(Cv)_i = (Dv)_i$, this implies that $((C-D)v)_i = 0$. But $((C-D)v)_i$ is simply the inner product $\langle C_i - D_i, v\rangle$. As $(C_i - D_i)$ is a non-zero vector, this inner product is uniformly distributed over $\mathbb{F}$[1]. Thus, the probability that $((C-D)v)_i = 0$ is at most $1/|\mathbb{F}|$. $\square$

5 Interaction in Proofs

In an *interactive proof system* [GMR89, Bab85], there are two parties – a computationally unbounded prover P, and a computationally efficient verifier V. The prover tries to convince the verifier that a given statement is true. In order to do this, the parties interact with each other according to a pre-specified protocol, at the end of which the verifier decides to either accept or reject. The algorithms followed by the verifier V and prover P are specified separately, and we denote by (P, V) the protocol where V interacts with P. As before, the statements we are interested in will be of the form $x \in L$ for some language L and input x. We will think of x as being the "input" that is given to both P and V before they start interacting.

We would like to design the protocol that P and V follow in such a way that when $x \in L$ is true, the verifier V accepts at the end of the protocol. And when $x \in L$ is not true, V should reject, even if the prover P cheats and deviates from the protocol specification. We usually denote a "cheating" prover that deviates from the protocol by P^*, to indicate that its behaviour is different from that of the "honest" prover P.

Definition 5.1. An interactive protocol (P, V), where V runs in polynomial time and is possibly randomised, is said to be and *interactive proof (IP) for a language L* if:

- **Completeness:** For every $x \in L$, when (P, V) is run with x as input, V accepts at the end with probability at least $2/3$.
- **Soundness:** For every $x \notin L$, and any prover strategy P^*, when (P^*, V) is run with x as input, V accepts at the end with probability at most $1/3$.

Most of our discussions in this course will be built upon this model of interactive proofs. To start with, let us ask the question of which languages L have such IPs. It is immediately clear that any language in NP or MA has an IP, where the prover just sends the verifier the witness y corresponding to any $x \in L$. But what about languages outside MA? Next, we will see an example of such a language that has an IP.

5.1 Graph Non-Isomorphism

Recall that a graph is specified by a set of vertices V and a set of edges E that go between them. We will represent a graph $G = (V, E)$ on n vertices using its adjacency matrix $A^G \in \{0,1\}^{n \times n}$, where the coordinate $A^G_{ij} = 1$ iff $(i, j) \in E$. Note that here, without loss of generality, the vertices of G are labelled by elements of $\{1, 2, \ldots, n\}$ (denoted by $[n]$).

A *relabelling* of a graph $G = (V, E)$ is a new graph $G' = (V, E')$ that is derived from G using a relabelling permutation $R : V \to V$ as follows: for every (u, v) that is contained in E, $(R(u), R(v))$ is contained in E'. In other words, G' is obtained from G by simply renaming the vertices in G.

Exercise 3. *Given that we represent graphs using adjacency matrices, what does applying a relabelling to a graph look like? That is, given the adjacency matrix A^G for a graph G and a relabelling permutation $R : [n] \to [n]$, how does one efficiently compute the adjacency matrix of $R(G)$?*

Definition 5.2. Two graphs G_0 and G_1 are said to be isomorphic if one is a relabelling of the other.

[1]Proof of this statement left as exercise.

The language we will be concerned with asks whether two given graphs are isomorphic or not.

Definition 5.3. The language GNI consists of all pairs of graphs (G_0, G_1) that have the same number of vertices and are *not* isomorphic to each other.

The problem of determining whether two given graphs are isomorphic or not is one that has been studied for a very long time, and is not known to have a polynomial-time algorithm. The best known algorithm was discovered quite recently by Babai [Bab16], and runs in time $2^{\log n^{O(1)}}$ for graphs on n vertices. It is easy to see that the GNI problem is in coNP, as the fact that two graphs are isomorphic can be proven by presenting the relabelling under which they are the same. The problem, however, is not known to be contained in NP or in MA.

We will construct a simple interactive proof using which the prover can prove to the verifier that two given graphs are not isomorphic. This protocol works as follows on input (G_0, G_1), where each graph is on n vertices:

- Verifier:
  1. Sample a uniformly random relabelling permutation $R : [n] \to [n]$, and a uniformly random bit $b \leftarrow \{0, 1\}$.
  2. Send $R(G_b)$ to the prover.
  3. When the prover responds with bit b', accept if $b = b'$, and reject otherwise.
- Prover:
  1. Upon receiving a graph G' from the verifier, set the bit $b' = 0$ if G' is isomorphic to G_0, and $b' = 1$ otherwise.
  2. Send b' to the verifier.

It is easy to verify that the verifier's algorithm above may be implemented in polynomial time. We need to prove now that the protocol is complete and sound.

Completeness: If the graphs G_0 and G_1 are not isomorphic, then the graph $R(G_b)$ can only be isomorpic to one of them, as isomorphism is a transitive property. Thus, b' is always computed to be the same as b by the prover, and the verifier always accepts. Thus, the protocol is perfectly complete.

Soundness: Suppose the graphs G_0 and G_1 are isomorphic. Then, there exists a permutation R_0 such that $R_0(G_0) = G_1$.

Claim 5.1. *For any G', the number of relabelling permutations R such that $R(G_0) = G'$ is the same as the number of permutations R such that $R(G_1) = G'$.*

Proof. Fix any G'. Given any R such that $R(G_1) = G'$, note that $R(R_0(G_0)) = G'$. Further, given any R' such that $R'(G_0) = G'$, we have $R'(R_0^{-1}(G_1)) = G'$. Thus, we can define a bijection between the set of R's that map G_1 to G' and the set of R's that map G_0 to G'. This bijection takes R to $R \circ R_0$ (where $\circ$ denotes composition), and its inverse takes R to $R \circ R_0^{-1}$. This implies that both these sets are of the same size. □

Claim 5.1 implies that the distribution of $R(G_0)$ and $R(G_1)$ are exactly the same when R is sampled uniformly at random. So, given just $R(G_b)$, the prover has no information about b whatsoever. Even if the prover deviates from the specified protocol, it has no way of making the verifier accept with probability more than $1/2$ (which it can do by guessing b at random). Thus, the protocol is sound with soundness error $1/2$.

This error can be made smaller by repeating the protocol sequentially. In later lectures, we will see more general ways of improving soundness and completeness errors.

Remark 5.1. *Note how, in this protocol, it was important that the verifier kept its randomness secret from the prover. That is, if the random b or information about R was known to the prover, it could quite easily break soundness. Such* private randomness *is often a useful tool in designing interactive proofs. We will see in later lectures, however, that this is not quite crucial in general.*

6 Limits on Interactive Proofs

So far, we have seen that any language in NP and MA has an interactive proof, and also an example of a language (potentially) outside these classes that does. How much more powerful are these proofs? Is there an interactive proof for any decidable language? How about languages that need an exponential amount of time to decide?

It turns out there is an upper bound on the power of these proofs in terms of a very natural complexity measure.

Definition 6.1. The class PSPACE consists of all languages that can be decided by an algorithm that uses a polynomial amount of space.

Theorem 6.1. $\mathsf{IP} \subseteq \mathsf{PSPACE}$

In other words, for any language L that has an interactive proof, there is also an algorithm that runs in polynomial space that can determine whether $x \in L$ for any x. We will prove a much weaker version of this theorem, noting that the entire theorem can be proven by extending the technique involved.

Theorem 6.2. $\mathsf{MA} \subseteq \mathsf{PSPACE}$

Note that an MA proof corresponds precisely to a 1-message IP.

Proof Sketch of Theorem 6.2. Suppose language $L \in \mathsf{MA}$. That is, there is a randomised polynomial-time verifier V such that for every x, there exists a polynomial-sized witness y such that $V(x, y)$ accepts with probability more than $2/3$ iff $x \in L$. Then, we show that $L \in \mathsf{PSPACE}$. We do this by showing the following two things:

1. There is a poly-space algorithm A that, for any $x \in L$, finds a y such that $V(x, y)$ accepts with probability at least $2/3$.
2. There is a poly-space algorithm B that computes, for any x and y, the probability that $V(x, y)$ accepts.

Both these algorithms are quite simple:

- $B(x, y)$ iterates through all possible random strings r that $V(x, y)$ could use, and counts the number of them under which $V(x, y; r)$ accepts.
- $A(x)$ iterates through all possible y's of the appropriate length (that is some polynomial in $|x|$), runs $B(x, y)$ for each, and remembers the y for which this value is the largest.

It is easy to see that both these algorithms only use polynomial space – note that V also can only use polynomial space as it runs in polynomial time, and that the same space can be reused for each execution of V by B or B by A.

Given these algorithms, for each x, membership in L can be determined by running $A(x)$ to find the optimal witness y, and checking whether $A(x, y) \geq 2/3$. $\square$

Exercise 4. *Extend the above proof to prove Theorem 6.1. Start by thinking about 2-message IPs. What would the algorithm A have to look for in place of the witness y here?*

References

[AW21] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 522–539. SIAM, 2021.

[Bab85] László Babai. Trading group theory for randomness. In Robert Sedgewick, editor, *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 421–429. ACM, 1985.

[Bab16] László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 684–697. ACM, 2016.

[Fre77] Rusins Freivalds. Probabilistic machines can use less running time. In Bruce Gilchrist, editor, *Information Processing, Proceedings of the 7th IFIP Congress 1977, Toronto, Canada, August 8-12, 1977*, pages 839–842. North-Holland, 1977.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.