INTERACTIVE PROOF SYSTEMS

EZRA FURTADO-TIWARI

Abstract.
In this paper, we look at the potential of the class **IP** of interactive proofs. We first look at deterministic verifiers/provers, and then continue on to survey the class **IP**, which includes nondeterministic verifiers and provers. We go on to show **IP** = **PSPACE**, and then show some examples of problems that are believed to not be in **NP** but that we have interactive proofs for.

1. Introduction

Interactive proof systems are a complexity class in which we allow two machines to cooperate with each other in order to solve problems: we call these machines the prover and the verifier, respectively. We consider nondeterministic machines for both, such that the system accepts solutions with high probability, and rejects non-solutions with high probability. Interactive proof systems were first studied in the 1950s, but continue to be studied (especially due to their use in cryptography).

In this paper, we will show that interactive proof systems are at least as powerful as the more well known classes **P** and **NP**. We will discuss a proof that **IP** = **PSPACE** later in the paper; **PSPACE** is conjectured to be more powerful than **P**, so we might conjecture that interactive proof systems are also more powerful than **P**.

2. Deterministic Interactive Proof Systems

We start by considering more restricted interactive proof systems; we will show that simply adding interaction does not give us an advantage in terms of the problems we can solve.

Definition 2.1. Let $f, g : \{0,1\}^* \to \{0,1\}^*$ be functions. *A k-round interaction* between f and g on an input $x \in \{0,1\}^*$ is a sequence $a_1, a_2, \ldots, a_k$ such that

$$\begin{aligned} a_1 &= f(x) \\ a_2 &= g(x, a_1) \\ a_3 &= f(x, a_1, a_2) \\ &\cdots \\ a_{2n} &= g(x, a_1, \ldots, a_{2n}) \\ a_{2n+1} &= f(x, a_1, \ldots, a_{2n+1}). \end{aligned}$$

We denote this interaction by $\langle f, g \rangle (x)$. We let the *output* of f after interaction with g be $f(x, a_1, \ldots, a_k)$, and denote this by $\mathbf{out}_f \langle f, g \rangle (x)$.

Date: December 11, 2022.

Essentially, we have two machines communicating by giving each other inputs based on what they receive. Note that we do not strictly follow the definition of a 'function' here, but instead assume that given a series of arguments, the 'functions' in question will provide exactly one output.

Definition 2.2. A language L has a *k-round deterministic interactive proof system* if there exists a deterministic Turing Machine V such that when given $(x, a_1, \ldots, a_k)$, it runs in polynomial time in $|x|$ and satisfies the following properties:

$$x \in L \implies \exists Q : \{0,1\}^* \to \{0,1\}^* \mathbf{out}_V \langle V, Q\rangle (x) = 1$$
$$x \notin L \implies \forall Q : \{0,1\}^* \to \{0,1\}^* \mathbf{out}_V \langle V, Q\rangle (x) = 0.$$

We call these properties completeness and soundness, respectively. Last, we define the class **dIP** to be the class consisting of $k(x)$-round deterministic interactive proof systems, where $k(x)$ is a polynomial in x.

Note that Q provides a proof, while V verifies it. In other words, the first statement means that every true statement has a proof, while the second statement means that no false statements have proofs.

Theorem 2.3. dIP = NP.

Proof. Note first that every language in **NP** can be verified in a single round of interaction. Thus, it suffices to prove that every language $L \in$ **dIP** is also in **NP**. Note that to certify that $L \in$ **NP**, we just need to verify the certificate $(a_1, a_2, \ldots, a_k)$. Then we just need to check that a verifier V exists such that $V(x) = a_1, V(x, a_1, a_2) = a_3, V(x, a_1, a_2, a_3, a_4) = a_5, and V_(x, a_1, \ldots, a_k) = 1$, as we can then construct a prover P such that $P(x, a_1) = a_2, P(x, a_1, a_2, a_3) = a_4, P(x, a_1, \ldots, a_{k-1}) = a_k$. Thus $\mathbf{out}_V \langle V, P\rangle (x) = 1$, so $x \in L$ and thus every language $L \in$ **dIP** is also in **NP**. ■

Essentially, using interactive proofs without randomness does not increase our options at all, which is surprising. We might wonder whether nondeterministic interactive proofs are better, so we consider this class next.

3. Nondeterministic Interactive Proof Systems

Definition 3.1. Let $z : \mathbb{N} \to \mathbb{N}$ be a function such that $z(n)$ is computable in polynomial time. A language L is considered to be in $\mathbf{IP}[z]$ if there exists a Turing Machine V such that when given $(x, a_1, \ldots, a_k)$, it runs in polynomial time in $|x|$ and satisfies the following properties:

$$x \in L \implies \exists Q : \{0,1\}^* \to \{0,1\}^* P\left[\mathbf{out}_V \langle V, Q\rangle (x) = 1\right] \geq \frac{2}{3}$$
$$x \notin L \implies \forall Q : \{0,1\}^* \to \{0,1\}^* P\left[\mathbf{out}_V \langle V, Q\rangle (x) = 1\right] \leq \frac{1}{3}.$$

Again, we refer to these properties as completeness and soundness, respectively. Last, we define the class **IP** to be $\cup_{c=1}^{\infty} \mathbf{IP}\left[n^c\right]$.

In other words, all $L \in \mathbf{IP}[z]$ have proofs with probability $\geq \frac{2}{3}$, and the chance that there exists a proof for a language $L \notin \mathbf{IP}[z]$ is $\leq \frac{1}{3}$. We note here that $\frac{2}{3}$ and $\frac{1}{3}$ are not special numbers and can be replaced by different values x_1 and x_2, respectively (both less than 1, satisfying $x_1 + x_2 = 1$) without changing the power of the proof system.

Here we discover a surprising result; adding randomness to our interaction makes interactive proofs quite a bit more powerful.

Theorem 3.2. $\mathbf{IP} = \mathbf{PSPACE}$.

First, we can prove $\mathbf{IP} \subseteq \mathbf{PSPACE}$:

Lemma 3.3. $\mathbf{IP} \subseteq \mathbf{PSPACE}$.

Proof Sketch. Note that given any verifier, we can compute every possible path of interaction between it and a prover, and construct the optimal prover in polynomial space (and $2^{\text{poly}|x|}$ time). Thus $\mathbf{IP} \subseteq \mathbf{PSPACE}$. ■

It remains to prove that $\mathbf{PSPACE} \subseteq \mathbf{IP}$. To prove this, we start by considering the problem TQBF.

Definition 3.4. A *quantified boolean formula* is a boolean formula in the form

$$Q_1 x_1 Q_2 x_2 Q_3 x_3 \dots Q_n x_n \varphi(x_1, x_2, \dots, x_n),$$

where $\varphi(x_1, x_2, \dots, x_n)$ is an unquantified boolean statement, the x_i are variables, and the Q_i are *quantifiers* ($\forall$ or $\exists$). We define TQBF to be the set of true quantified boolean formulae.

Lemma 3.5. $\mathsf{TQBF} \in \mathbf{PSPACE}$.

Proof. Let

$$\psi = Q_1 x_1 Q_2 x_2 Q_3 x_3 \dots Q_n x_n \varphi(x_1, x_2, \dots, x_n)$$

be a quantified boolean formula, and let the sizes of ψ and $\varphi(x_1, x_2, \dots, x_n)$ be n and m, respectively. First, note that we can assume $n > 0$, as a formula with no variables contains only $\varphi(c_1, c_2, \dots)$ for some constants c_i, and can thus be evaluated with $O(m)$ space. For $n > 0$, consider the following recursive algorithm X:

(1) Let the process $\psi_{x_1|c}$ represent the statement replacing x_1 with the constant c and removing the quantifier Q_1.
(2) Evaluate $X(\psi_{x_1|c})$ and $X(\psi_{x_2|c})$.
(3) If $Q_1 = \exists$, then check that one of the two calls to the algorithm returns 1; otherwise check that both return 1. If this is satisfied, return 1; else return 0.

The trick here is to store the results of computation efficiently. Consider both calls to X (note that there are 2 for each $n \geq 0$). We can first compute the result of one computation, store it, and then use the same space the compute the result of the other computation. Thus we use $O(n) + O(n \cdot m) = O(n \cdot m)$ space, so $\mathsf{TQBF} \in \mathbf{PSPACE}$. ■

Lemma 3.6. TQBF is **PSPACE**-complete.

We omit this proof from this paper; the reader is encouraged to consult [AB07] for the proof.

Lemma 3.7. $\mathbf{PSPACE} \subseteq \mathbf{IP}$.

Proof. We first consider the process of arithmetization. Essentially, we can convert our boolean formulae into polynomials. Let variables X, Y, Z represent booleans x, y, z (so that they take on values 0 and 1). Then we can make the following reductions:

$$\begin{aligned} x \wedge y &\iff X \cdot Y \\ \neg x &\iff (1 - X) \\ x \vee y &\iff \neg(\neg x \wedge \neg y) \iff 1 - (1 - X)(1 - Y). \end{aligned}$$

Then for each statement $\varphi(x_1, x_2, \ldots, x_n)$, we can consider an equivalent polynomial $P_\varphi(X_1, X_2, \ldots, X_n)$.

Consider a specific quantified boolean formula

$$\sigma = \exists x_1 \forall x_2 \exists x_3 \ldots \forall x_n \varphi(x_1, x_2, \ldots, x_n).$$

Then the arithmetization of this formula is

$$\sum_{X_1 \in \{0,1\}} \prod_{X_2 \in \{0,1\}} \sum_{X_3 \in \{0,1\}} \cdots \prod_{X_n \in \{0,1\}} P_\varphi(X_1, X_2, \ldots, X_n),$$

and we need to check that this is greater than 0. Now, note that since all X_i are either 0 or 1, we can convert our polynomial $P_\varphi(X_1, X_2, \ldots, X_n)$ into an equivalent multilinear polynomial $K_\varphi(X_1, X_2, \ldots, X_n)$, where the degree of each term is 1. This would allow us to reduce our problem to a sum

$$\sum_{X_1 \in \{0,1\}} \sum_{X_2 \in \{0,1\}} \sum_{X_3 \in \{0,1\}} \cdots \sum_{X_n \in \{0,1\}} g(X_1, X_2, \ldots, X_n)$$

for some polynomial g. We can use a similar algorithm to the one we used to prove $\mathsf{TQBF} \in$ **PSPACE** to compute this using an interactive proof.

Consider the following protocol:

(1) (Prover) If $n = 1$, check that $g(0) + g(1) > 0$. If so, accept, else reject. Otherwise, ask the verifier to send a polynomial $f(a)$, where
$$f(a) = g(a, X_2, \ldots, X_n).$$
(2) (Verifier) Send the polynomial $f(a)$.
(3) (Prover) If $f(0)+f(1) = 0$, then reject. Otherwise, pick a random b. Use this protocol to check
$$0 < g(b, X_2, \ldots, X_n).$$

Note that since TQBF is **PSPACE**-complete, we can conclude that **PSPACE** $\subseteq$ **IP**. ■

Proof of Theorem 3.2. Combining Lemma 3.3 and Lemma 3.7, we have that **IP** = **PSPACE** as desired. ■

4. Graph Non-Isomorphism

In this section we attempt to provide an example of an interactive proof, while also showing the strengths of this proof method. We know quite a bit about the graph isomorphism problem, notably that it is in **NP**. Because it is in **NP**, we know that it is in **IP**. Since **IP** = **PSPACE** and **PSPACE** is closed under complements, we know that the complement of graph isomorphism is in **IP**.

Consider two graphs G and H, and assume that we wish to prove that $G \ncong H$. We consider the following protocol to solve this problem:

(1) (Verifier) Generate some graph G' isomorphic to G, and some graph H' isomorphic to H.
(2) (Verifier) Decide randomly (using a private coin flip) whether or not to send (G, G') or (G, H').
(3) (Prover) Send the result of the coin flip, encoded as a bit.
(4) (Verifier) If the prover is wrong about the result, then reject. Otherwise, continue.
(5) Repeat a constant number of times.

If the prover is wrong about the result, then we determine that the prover is unable to tell the difference between the two graphs, and thus we reject. Otherwise, we assume that the prover is better than guessing, and thus can determine whether or not the two graphs are isomorphic. Using a large number of repetitions, we can ensure that the prover is correct about the result a large proportion of the time, so this is enough to satisfy that the protocol is a valid interactive proof.

Similarly, we note that an interactive proof can be formulated for the complement of any problem in **NP**. In other words, **co-NP** $\subseteq$ **PSPACE** $=$ **IP**.

5. Conclusion

Interactive proofs allow us to solve a much wider range of problems than we know to be in **NP**. Showing that **IP** $=$ **PSPACE** allows us to further formalize which kinds of problems we can solve using interactive proof systems, and also allows us to understand the types of problems that we would be able to solve if **co-NP** $=$ **NP**, **P** $=$ **NP**, **PSPACE** $=$ **NP**, and so on. However, there are still many interesting questions relating to interactive proof systems alone.

For instance, when we described a protocol for graph nonisomorphism, it was important to the protocol that the coin was flipped in private. But how would the class change if we were forced to use only public coins? This gives us the class **AM** of Arthur-Merlin games. An important result is that **AM** $=$ **IP** (see [Lau11]).

In addition to this, we can restrict the output that can be given by the prover and verifier, which would give us yet another system: zero-knowledge proofs (**ZKP**). Notably, **NP** $\subseteq$ **ZKP** (see [Zul06]), which gives us more information about the power of the class **IP**. Even with restricted machines, the possibility of randomness allows us to possibly solve more problems than can be solved without interaction.

Various problems still remain open in both of these fields.

References

[AB07] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2007.
[Lau11] Peeter Laud. Complexity theory. 2011.
[Zul06] Florian Zuleger. Interactive proof systems. 2006.