1 Practice Problems

Solutions are provide on the following page.

1.1 Polynomial Multiplication

Consider the following language

\[ R = \{ \langle p, q, r \rangle \mid p, q, r \text{ are polynomials with integer coefficients such that } p(x)q(x) = r(x) \} \]

Let \( d = \max(\deg p, \deg q) \), the maximum degree between \( p \) and \( q \). It follows that \( R \in \mathbf{P} \) since the FOIL method to multiply polynomials takes \( O(d^2) \) multiplications.\(^1\) Derive a randomized algorithm that uses \( O(d) \) multiplications to decide \( R \) and returns the correct answer w/ probability \( \frac{2}{3} \). How can we modify the algorithm to return the correct answer with probability \( 1 - \epsilon \)? How many multiplications does this take in terms of \( d \) and \( \epsilon \)?

1.2 Certificates and Oracles

We define \( \mathbf{NP}^C = \bigcup_{A \in C} \mathbf{NP}^A \) for a complexity class \( C \). In other words it is the set of languages decidable in polynomial time by a non-deterministic Turing Machine with an oracle that can determine whether or not \( x \in A \) for any \( A \in C \) i.e. \( \mathbf{NP}^C \) has access to every oracle that queries a language in \( C \). Prove \( \mathbf{NP}^\mathbf{NP} \cap \mathbf{coNP} = \mathbf{NP} \).

\(^1\) In fact this can be done using \( O(d \log d) \) multiplications with the Fast Fourier Transform
2 Solutions

2.1 Polynomial Multiplication

The idea is that instead of multiplying out the polynomials we can randomly evaluate them and check for equality. If \( p(x)q(x) = r(x) \), an evaluation will always agree with the equality so we will never return a wrong answer. On the other hand if \( p(x)q(x) \neq r(x) \) then there are at most \( 2d \) evaluations which would disagree since the polynomial \( s(x) = p(x)q(x) - r(x) \) has \( 2d \) roots by the Fundamental Theorem of Algebra. Therefore the random evaluation is selected from a large enough space so that it is unlikely to select a root. Since we can evaluate the polynomials with \( O(d) \) multiplications, this leads to the following algorithm

\[ M : \text{"On input } \langle p, q, r \rangle \]

1. Choose a random number \( y \in \{1, 2, \ldots, 6d\} \)
2. If \( p(y)q(y) = r(y) \), accept.
3. Else, reject.

When \( \langle p, q, r \rangle \in R \) we return the correct answer with probability 1. If \( \langle p, q, r \rangle \notin R \) then we return the correct answer with probability at least \( \frac{4d}{6d} = \frac{2}{3} \) as desired.

To decrease the probability of error, we can repeat the above algorithm multiple times and accept if and only if all repetitions return accept. Since the probability of error for one repetition is at most \( \frac{1}{3} \), the probability of error for \( k \) repetitions is at most \( \left( \frac{1}{3} \right)^k \). By setting \( k = \log_3 \left( \frac{1}{\epsilon} \right) \) we get at most \( \epsilon \) error with \( O(d \log(1/\epsilon)) \) multiplications.

Note: Some of you may notice that choosing random \( y \in \mathbb{R} \) would reduce the probability of error to 0. However this is not valid since our model only supports randomness through polynomial many fair coin tosses which is why we took a finite set.

2.2 Certificates and Oracles

We show \( \text{NP} \cap \text{coNP} \subseteq \text{NP} \). This can be done by converting any oracle query to \( \text{NP} \cap \text{coNP} \) to an appropriate certificate. Suppose our \( \text{NP} \) algorithm would like to query \( x \in A \) where \( A \in \text{NP} \cap \text{coNP} \). Instead of using the oracle, we can provide the \( \text{NP} \) certificate if \( x \in A \) and the \( \text{coNP} \) certificate if \( x \notin A \). Formally our new \( \text{NP} \) certificate would be of the form \( \langle v, y_1q_1, y_2q_2, \ldots \rangle \) where \( v \) is the original certificate, \( y_i \in \{0,1\} \) indicates if \( x_i \in A_i \) or \( x_i \notin A_i \), and \( q_i \) is the corresponding certificate for the \( i \)th query to the oracle.