CMPE 300: Analysis of Algorithms

Project 3 - Interactive Proof Systems

Yasin Başтuğ *
Erencem Özbey †

Deadline: 11 January 2026, 23:59

– TUTORIAL –

1- Turing Machine

A Turing machine is a mathematical model of computation that executes an algorithm in a step-by-step, sequential way. Informally a Turing machine consists of:

- A 1D tape divided into cells, which serves as memory.
- A single read/write head that moves one cell to the left or right at each step.

Key points for this project:

- Computation is sequential: one step after another.
- Memory on the tape is infinite and accessed in a non random–access fashion: the head moves locally, one cell at a time.
- Despite this restricted access pattern, Turing machines can implement any algorithm that can be executed by a realistic computer (this is the idea of Turing completeness).

Throughout the complexity theory, the running time of an algorithm is defined as the number of steps a Turing machine uses as a function of the input size.

*osman.bastug@std.bogazici.edu.tr
†erencem.ozbey@std.bogazici.edu.tr

2- Complexity Classes: P and NP

We first need to define what a deterministic Turing machine is to explain this concept in a better way. A deterministic Turing machine can't operate in a parallel way. However a nondeterministic Turing machine can operate. For example you can check Hamiltonian paths easily with a Nondeterministic Turing machine because you can take a look at all node permutations in a parallel way.

Class P

The class P consists of all decision problems that can be solved in polynomial time by a deterministic Turing machine. That is, there exists an algorithm (TM) that, on input of size n, halts with the correct YES/NO answer in time at most n^k for some constant k.
Intuitively:

- Problems in P are considered efficiently solvable.
- Example: shortest paths in graphs

Class NP

The class NP consists of all decision problems that can be solved in polynomial time by a "nondeterministic" Turing machine. That is, there exists an algorithm that, on input of size n, halts with the correct YES/NO answer in time at most n^k for some constant k.

- Examples: Hamiltonian Path, 3-coloring

Equivalently, NP consists of decision problems for which:

- If the answer is YES, then there exists a witness (or certificate) that can be verified in polynomial time by a deterministic Turing machine. If you give me a Hamiltonian path it would be really easy to check if it is true or not.

3 Probabilistic Algorithms

A probabilistic algorithm is an algorithm whose behavior may vary from run to run on the same input. For such algorithms we typically analyze the expected running time and probability of correctness: often, we require that the algorithm answers correctly with high probability.
Examples: Randomized Quicksort, Freivald's algorithm (verifying matrix multiplication), randomized primality tests (e.g. Miller-Rabin) test

4 Interactive Proof Systems

An interactive proof system[1] is a protocol between two parties:

- A powerful Prover P, who knows something and wants to convince the verifier that a statement is true.
- A Verifier V, who wants to be convinced but does not fully trust the prover.

Verifier uses randomness and probabilistic challenges, and tries to avoid cheaters by asking questions. They exchange messages for several rounds, after which the verifier either accepts or rejects the statement.
An interactive proof should satisfy:

- Completeness: If the statement is true and P is honest, then V accepts with high probability.
- Soundness: If the statement is false, then no (even cheating) prover P can make V accept with high probability.

In zero-knowledge interactive proofs, we additionally require:

- Zero–knowledge: The verifier learns nothing beyond the truth of the statement.

Randomness is essential:

- The verifier uses random challenges.
- The prover often uses randomness to hide its secret witness.
- Soundness and zero–knowledge are analyzed using probability.

–QUESTIONS–

This project guides you through several topics and problems including 3–coloring, graph isomorphism, NP-completeness, and interactive proofs. You are asked not only to describe protocols, but also to reason about why they work, especially from the probabilistic point of view.

Q1) Interactive Algorithm for 3–Coloring (50 pt)

a) Let $G = (V, E)$ be a graph. Explain what it means for G to be 3–colorable. Define the decision problem: $3\text{COLOR} = \{G : G \text{ is 3–colorable}\}$.

b) Describe a zero-knowledge interactive proof protocol in which:

- The Prover knows a valid 3–coloring of G.
- The Verifier wants to be convinced that G is 3–colorable, but does not learn the actual coloring.

[1] `https://en.wikipedia.org/wiki/Interactive_proof_system`

c) Probabilistic analysis: Explain why this protocol works:

- Argue completeness: why does an honest prover make the verifier accept with high probability?
- Argue soundness: if G is not 3-colorable, why can no cheating prover convince the verifier with more than a small probability? How does repeating the protocol reduce this probability?
- Discuss briefly (at a high level) why the verifier learns nothing about the specific coloring essentially. (Zero–knowledge intuition.) Emphasize the role of randomness and probabilities in your explanation.

Q2) Interactive Algorithm for Graph Isomorphism (50 pt)

a) Define the Graph Isomorphism problem: $GI = \{(G_1, G_2) : G_1 \sim= G_2\}$, where $G_1 \sim= G_2$ means there exists a bijection of vertices that preserves edges.

b) Describe a zero-knowledge interactive protocol where:

- The Prover knows an isomorphism $\pi : G_1 \to G_2$.
- The Verifier wants to be convinced that G_1 and G_2 are isomorphic, without learning π.

c) Probabilistic analysis: Explain again why this protocol works:

- Completeness: why an honest prover convinces the verifier.
- Soundness: if G_1 and G_2 are not isomorphic, why does any cheating prover have at most a certain probability of success per round? How is this probability reduced by repetition?
- Zero–knowledge intuition: why do the random permutations and responses prevent the verifier from learning the specific isomorphism π?

– BONUS PART 1 –

Q3) Trick the Verifier for 3–Coloring (20 pt)

In this bonus, you will study the *soundness* of an interactive proof for graph 3–coloring and understand why the verifier's randomness plays a critical role.

We will first describe how the 3–coloring interactive proof is normally defined in theory (using cryptographic commitments), and then we will analyze a simplified implementation in which verifier bias can be exploited by carefully chosen inputs.

This demo[2] can help you comprehend the protocol visually.

[2] `https://web.mit.edu/~ezyang/Public/graph/svg.html`

Background: The 3–Coloring Interactive Proof

Let $G = (V, E)$ be a graph. In the standard zero–knowledge interactive proof for 3–coloring:

- The prover knows a valid 3–coloring $c : V \to \{0, 1, 2\}$.
- The verifier wants to be convinced that such a coloring exists, without learning the coloring itself.

At a high level, the protocol works as follows:

1. The prover chooses a random permutation π of the colors $\{0, 1, 2\}$ and defines $c'(v) = \pi(c(v))$.
2. The prover uses a commitment scheme to hide and bind every node.
3. The verifier selects a random edge $(u, v) \in E$.
4. The prover opens the commitments for u and v, and the verifier checks if the two revealed colors are different.

Repeating this interaction reduces the soundness error. Commitments force the prover to fix a coloring *before* the verifier reveals which edge it will check.

Simplified Setting Used in This Assignment

To keep the programming component focused on probabilistic reasoning rather than cryptography, the provided code implements a *simplified* version of the protocol:

- Cryptographic commitments are *not* implemented in code.
- In each round, the verifier challenges an edge (u, v).
- The prover responds directly (without lying) with two colors $c_u, c_v \in \{0, 1, 2\}$.
- The verifier accepts the round if $c_u \neq c_v$.

The verifier repeats this test multiple times and accepts if all rounds succeed. Importantly, the verifier does *not* choose edges uniformly at random.

What We Provide

We provide the following components:

- `gen_graph`: an executable that generates non–3–colorable graphs.
- `zk_verifier`: an executable verifier that repeatedly challenges edges according to a biased distribution.
- `honest_prover.py`: a prover that answers *consistently* according to a fixed coloring.

You are not expected modify the internals of the graph generator or the verifier.

Generating Graphs

To generate a graph of size N, run:

```
./gen_graph --size N > G.txt
```

The output file `G.txt` contains:

- the number of vertices n,
- the number of edges m,
- followed by a list of m edges.

The generator should be treated as a black box.

Coloring File Format

For each graph, you must provide a coloring file.

The coloring file consists of exactly n lines, where line i contains the color assigned to vertex i.

- Colors must be integers in $\{0, 1, 2\}$.
- Vertex numbering starts from 0.

Example. For a graph with $n = 5$, a coloring file could be:

```
0
1
2
1
0
```

This assigns color 0 to vertex 0, color 1 to vertex 1, and so on.

Testing a Coloring

To test a graph–coloring pair, use the provided honest prover together with the verifier.

Assuming:

- `G.txt` is a graph generated by `gen_graph`,
- `coloring.txt` is your proposed coloring,

run:

```
python3 honest_prover.py \
  --verifier "./zk_verifier --graph G.txt --rounds 200" \
  --graph G.txt \
  --coloring coloring.txt
```

Because the verifier is probabilistic, you should run this experiment multiple times and **report the acceptance rate**.

Your Task

Your goal is to demonstrate that the verifier can be fooled *without modifying the protocol*, purely by choosing adversarial inputs.

Specifically, you must:

1. Generate graphs using `gen_graph`.
2. For each graph, construct a coloring that is *not* a valid 3–coloring, but violates only a small number of edges.
3. Test each graph–coloring pair using the provided verifier and `honest_prover.py`.
4. Identify instances where the verifier accepts with high probability despite the coloring being invalid.
5. Report your acceptance rates and explain why this behavior does not contradict the theoretical guarantees of the full protocol.

Important. You are not asked to write a cheating prover. All cheating must come from your choice of graphs and colorings. You must generate **20** graph files with gen_graph, pay attention to the sizes, analyze how it effects acceptance. Do not exceed 1000.

– BONUS PART 2 –

Q4) (5 pt) Describe this procedure to build zero-knowledge proofs. Your explanation should touch upon:

- How to reduce an NP problem to an NP–complete problem.
- How does the prover hide the secret answer commitments or other cryptographic tools are used to hide the witness while allowing consistent openings.
- Why this methodology is fundamentally probabilistic, and why the resulting interactive algorithm is expected to be sound: a cheating prover fails with high probability

Q5) (5 pt) Using the methodology from (Q4), outline how to construct an interactive proof for Hamiltonian Path. Explain why this interactive protocol achieves:

- Completeness: An honest prover with a Hamiltonian path convinces the verifier with high probability.
- Soundness: If there is no Hamiltonian path, any cheating prover can only succeed with small probability per round.
- Zero–knowledge intuition: The verifier does not learn the actual Hamiltonian path in the original graph.

Be explicit about where randomness is used and how the probabilities are analyzed. In particular, answer: Why is this probabilistic interactive algorithm going to work?

Submission Format

Submit a single ZIP archive named: `StudentNo1_StudentNo2.zip`.
Reports should be compiled in LaTeX. Include your bonus explanations in the Report.pdf.
If you solved Bonus Q3, include graphs and coloring folders in your submission.

Submissions must have the following layout:

```
StudentNo1_StudentNo2.zip
-- StudentNo1_StudentNo2/
    |-- Report.pdf
    |-- graphs/
        |--G_1.txt
        |--G_2.txt
        ...
    |-- colorings/
        |--coloring_1.txt
        |--coloring_2.txt
        ...
```

Notice 1: The deadline is strict. No late submissions will be accepted.

Notice 2: Use of Large Language Models (LLMs) is strictly prohibited.