CS5234 Project: Streaming Interactive Proofs

Nguyen Dang Phuc Nhat (A0184583U), Anju Gopalkrishnan (A0254372E),
Grace Ngu Sook Ern (A0268281X)

1 Introduction

The increasing popularity of commercial cloud computing services, and more generally outsourced computations, has revealed compelling new applications for exploring interactive proofs with highly restricted clients. Imagine a computationally weak user (Agency/Business) utilizing a commercial cloud computing service for data storage and processing. In this setup, the user can send data-related queries and cloud responds with the answer. However, due to the untrusted nature of the cloud, the user seeks assurance regarding the accuracy of the responses. Consequently, the user initiates interrogating the cloud by sending a series of challenges to the cloud, receiving corresponding responses. At the end of this interaction, the user must decide whether to accept the provided answer as valid or reject it as invalid. Typically, the challenges are randomized to prevent the cloud from predicting the upcoming challenge in advance. This general scenario is referred as *verifiable data stream computation*. [Cha+15]

Streaming Interactive Proofs (SIP): A computationally limited client (Verifier) wants to compute some property of a massive stream of input, but lacks the resources to store even a small fraction of the input and so cannot perform the desired computation locally. The client therefore outsources the processing to a powerful but untrusted third party service provider (Prover), who can perform the requested computation and proves that the answer is correct via a short interation after the stream is seen. As per the setup for SIP communication classes, there are 3 parties involved: Alice the stream provider, Bob the Verifier, and Merlin the Prover. We will use these names and Verifier/Prover terms interchangably.

There are a lot of nuances in designing SIP protocols that we have promptly skipped over in this project: public/private coins, the order of Alice-Bob and Bob-Merlin interactions, etc. In this project, we have focused on exploring the basic algebraic techniques used in SIPs, and their applications in streaming verification protocols.

2 Preliminaries

2.1 Useful Definitions

Given a finite field $\mathbb{F}$:

Theorem 1 (Fundamental theorem of algebra) *Over any field, a non-zero polynomial of degree n has at most n roots.*

Theorem 2 *Define a line ℓ in $\mathbb{F}^b$ to be the range of a nonconstant affine function from $\mathbb{F}$ to $\mathbb{F}^b$. Then there are exactly $|\mathbb{F}|$ points on ℓ.*

Theorem 3 *Given 2 points a, b in $\mathbb{F}^b$, there is a unique line through them.*

Theorem 4 *Let $p \neq q$ be univariate polynomials of degree at most n. Then p and q agree on at most n inputs. Equivalently: $Pr_{r \in \mathbb{F}}[p(r) = q(r)] \leq \dfrac{n}{|\mathbb{F}|}$.*

Lemma 5 *The Schwartz-Zippel Lemma is a multivariate generalization of Theorem 4. Let $p \neq q$ be l-variate polynomials of degree at most d. Then,* $\Pr_{r \in \mathbb{F}^l}[p(r) = q(r)] \leq \frac{d}{|\mathbb{F}|}$*. Total degree refers to the maximum sum of degrees of all variables in any term.*

2.2 Streaming Interactive Proofs - Formal definition

An interactive proof (IP) involves two parties: a Prover (P) and a Verifier (V). The Prover solves the problem and communicates the answer to the Verifier. In the subsequent conversation, the Prover tries to convince the Verifier of the correctness of the provided answer.
In the streaming setup, Verifier can process data from a "private" stream σ before or after interacting with Prover. In other words, Alice provides the stream σ to only Bob. Merlin can only interact with Bob, and is not necessarily aware of Alice.
Requirements. A Streaming Interactive Proof has to satisfy two requirements:

- Completeness: An honest P who follows the protocol perfectly will convince V to accept with high probability of at least $1 - \delta_c$.
- Soundness: If P lies and returns an invalid answer, then V will find out and reject with high probability of at least $1 - \delta_s$.

Usually in literature, a protocol is considered is valid if $\delta_c, \delta_s \leq 1/3$ [CTY11]. In our report, we will try to provide the analysis in terms with respect to δ_c, δ_s whenever possible.

2.3 `Equality` testing - Power of Randomness

This section gives a demonstration on the power of randomness in the context of a non-interactive problem `Equality`. We will then apply the main takeaways from this problem to designing SIP protocols.
In `Equality`, Alice and Bob holds binary strings $a, b \in \{0,1\}^n$ respectively. Bob needs to determine whether $a = b$ while minimizing the communication cost. A trivial solution: Alice can send a to Bob who checks whether $a = b$. Communication cost is exactly n bits. This solution is optimal amonngst deterministic protocols.
Randomized Protocol. Let $\mathbb{F}$ be any finite field with $|\mathbb{F}| \geq n^2$. Any bits in a, b can be seen as elements of $\mathbb{F}$. Alice and Bob will interpret their strings as the coefficients of the univariate polynomials $p(x)$ and $q(x)$ respectively, over the field $\mathbb{F}$. Let $p(x) = \sum_{i=1}^{n} a_i x^i$ and $q(x) = \sum_{i=1}^{n} b_i x^i$. Alice picks a random $r \in \mathbb{F}$ and sends $(r, p(r))$ to Bob. Bob outputs EQUAL if $p(r) = q(r)$. Otherwise he outputs NOT-EQUAL.
Analysis. If $a = b$, then Bob outputs EQUAL with probability 1. If $a \neq b$, then Bob outputs NOT-EQUAL with probability at least $1 - 1/n$ over the choice of $r \in \mathbb{F}$ (follows from Theorem 4). Total communication cost is $O(\log |F|) = O(\log n)$ bits for sending two field elements sent $(r, p(r))$.
Why is this useful?

- Corollary to Theorem 4: If two low-degree polynomials agree at a randomly chosen input, it is safe to believe that they are the **same** polynomial.
- Interpreting inputs as low-degree polynomials is powerful. If two inputs differ **at all**, then once interpreted as polynomials, they differ **almost everywhere**.

2.4 Multilinear Extension

In this section we discuss the Extension polynomials which is motivated from Lemma 5.

Definition 6 *Given a function $f : \{0,1\}^l \rightarrow \mathbb{F}$, a l-variate polynomial g over $\mathbb{F}$ is said to extend f if $f(x) = g(x)$ for all $x \in \{0,1\}^l$.*

One can think of an extension g of f as an error-corrected (or, at least, distance-amplifying) encoding of f in the following sense: if two Boolean functions f, f' disagree at even a single input, then any degree d extensions g, g'

must differ almost everywhere (assuming $d \ll |\mathbb{F}|$). This is made precise by the Lemma 5, which guarantees that g, g' agree on at most $d/|\mathbb{F}|$ points in $\mathbb{F}^v$. These distance-amplifying properties give the Verifier surprising power over the Prover.

Definition 7 *Any function $f : \{0,1\}^l \to \mathbb{F}$ has a unique multilinear extension (MLE), denoted by $\tilde{f} : \mathbb{F}^l \to \mathbb{F}$. Multilinear means the polynomial has degree at most 1 in each variable. More specifically, $\tilde{f}(z_1, z_2, ..., z_l) = \sum_{(i_1,i_2,...,i_l)\in\{0,1\}^l} f(i_1, i_2, ..., i_l)\chi_i(z_1, z_2, ..., z_l)$. The function χ_i acts as an identity function for binary inputs.*

	0	1
0	1	2
1	1	4

Figure 1: A function f mapping $\{0,1\}^2$ to the field $\mathbb{F}_5$.

	0	1	2	3	4
0	1	2	3	4	5
1	1	4	2	0	3
2	1	1	1	1	1
3	1	3	0	2	4
4	1	0	4	3	2

Figure 2: The multilinear extension, $\tilde{f}$ of f over $\mathbb{F}_5$.

3 Verification of Streaming Aggregation problems

3.1 $\texttt{Sum} - \texttt{check}$ protocol

$\texttt{Sum} - \texttt{check}$ protocol is a basic and efficient interactive proof protocol which can be applied to solve various *aggregation* problems. In this section, will discuss $\texttt{Sum} - \texttt{check}$ protocol with an oracle access for Verifier (adapted from the lecture notes of Thaler [Tha20a; Tha20b]). We will further explore how to modify this version to use in F_2 and F_k verification.

Protocol. V is given oracle access to a l-variate polynomial g over finite field $\mathbb{F}$. V will use this oracle and exchange l rounds of messages with P to verify the claim $H = \sum_{(b_1,b_2,...,b_l)\in\{0,1\}^l} g(b_1, b_2, ..., b_l)$.

In **Round 1**, P sends the univariate polynomial $s_1(X_1)$ claimed to be equal to $H_1(X_1) := \sum_{(b_2,...,b_l)\in\{0,1\}^{l-1}} g(X_1, b_2, ..., b_l)$. V checks and only proceeds if s_1 is an univariate polynomial of degree $\deg_1(g)$ and $H = s_1(0) + s_1(1)$. Note that $\deg_i(g)$ denotes degree of g in the $i-$th variable. If this check passes, it is safe for V to believe that H is the correct answer, as long as V has reasons to believe that $s_1 = H_1$. To check this, V can check if s_1 and H_1 agree at a random point r_1. V can compute $s_1(r_1)$ directly from P's first message but not $H_1(r_1)$. For this, V picks random element $r_1 \in \mathbb{F}$ and sends r_1 to P.

In **Round** $j < l$, P sends the univariate polynomial $s_j(X_j)$ claimed to be equal to $H_j(X_j) := \sum_{(b_{j+1},...,b_l)\in\{0,1\}^{l-j}} g(r_1, r_2, ..., r_{j-1}, X_j, b_{j+1}, ..., b_l)$. V checks and only proceeds if s_j is an univariate polynomial of degree $\deg_j(g)$ and $s_{j-1}(r_{j-1}) = s_j(0) + s_j(1)$. V picks random element r_j and sends to P. These rounds are equivalent recursive calls, where the polynomial to verify is $\sum_{(b_1,...,b_{j-1})\in\{0,1\}^{j-1}} g(b_1, ..., b_{j-1}, x_j, x_{j+1}, ..., x_l)$ and the claimed sum is $H_j(0) + H_j(1)$. By ignoring the non-variable parts of the polynomial, we can think of these rounds as repeatedly testing $s_1(r_1) = H_1(r_1)$.

In the last **Round** l, P sends an univariate polynomial $s_l(X_l)$ claimed to be equal to $H_l := g(r_1, ...r_{l-1}, X_l)$. V proceeds only if s_v is an univariate polynomial of degree at most $deg_l(g)$ and that $s_{l-1}(r_{l-1}) = s_l(0) + s_l(1)$. V picks r_l at random and checks if $s_l(r_l) = g(r_1, ...r_l)$, rejecting if not. No need for more rounds as V can perform this check with a single oracle query to g at $(r_1, r_2, ..., r_l)$. If V has not rejected yet, V halts accepts.

Correctness Analysis

- Completeness holds by design: If P sends the prescribed messages, then all of V's checks will pass.

- Soundness: If P is dishonest, then V rejects with probability at least $1 - \frac{l.d}{|\mathbb{F}|}$, where d is the maximum degree of g in any variable. Proof is by induction on the number of variables l:
  Base case: $l = 1$. In this case, P sends a single message $s_1(X_1)$ claimed to equal $g(X_1)$. V picks r_1 at random, checks that $s_1(r_1) = g(r_1)$. By Theorem 4, if $s_1 \neq g$, then $Pr_{r_1 \in \mathbb{F}}[s_1(r_1) = g(r_1)] \leq \frac{d}{\mathbb{F}}$.
  Inductive case: $l > 1$. V and P recursively performs checks akin to verifying $s_1(r_1) = H_1(r_1)$. By Theorem 4, if $s_1 \neq H_1$, then $\Pr_{r_1 \in \mathbb{F}}[s_1(r_1) = H_1(r_1)] \leq \frac{d}{\mathbb{F}}$. If $s_1 \neq H_1$, then P is left to prove a false claim in the recursive call. The recursive call applies sum-check to $g(r_1, X_2, ..., X_l)$, which is $l-1$ variate. By induction, P fails to convince V in the recursive call with probability at least $1 - \frac{l.d}{|\mathbb{F}|}$.
  Summary: If $s_1 \neq H_1$, then Probability V accepts is at most:
  $\Pr_{r_1 \in \mathbb{F}}[s_1(r_1) = H(r_1)] + Pr_{r_2,...r_l \in \mathbb{F}}[\text{V accepts}|s_1(r_1) \neq H(r_1)] \leq \frac{d}{|\mathbb{F}|} + \frac{d(l-1)}{|\mathbb{F}|} \leq \frac{d.l}{|\mathbb{F}|}$.

Analysis of space and communication cost. Total communication is $O(d.l)$ field elements. P sends l messages, each a univariate polynomial of degree at most d. V sends $l-1$ messages, each consisting of one field elements. The space used by the Verifier is bounded by keeping at most 2 polynomials for each round (V can even pre-compute $s_{j-1}(r_{j-1})$ and thus keep only the polynomial sent by P). A polynomial kept by V has at most $d+1$ coefficients which are field elements, so it needs $O(d+1)$ field elements, which can be simplified to $O(d \log(|\mathbb{F}|))$ bits.
Note that the sum-check protocol described above assumes that the Verifier has oracle access to g. However, this will not be the case in applications, as g will ultimately be a polynomial that depends on input stream.

3.2 Combination with Multilinear Extension

In the streaming setup, g has to be a function that can be evaluated at a random point efficiently using the stream. It is useful to compute $H = \sum x \in \{0,1\}^v f(x)$ for some function $f : \{0,1\}^v \to \mathbb{F}$ derived from the Verifier's input. We can verify H by applying the $\texttt{Sum}-\texttt{check}$ protocol to a specific extension g of f. Let $\tilde{f}$ denote the multilinear extension of f. Depending on the problem, we can consider $g = \tilde{f}$, or derived (efficiently) from $\tilde{f}$ in some way.
Despite being a powerful tool, $\tilde{f}$ cannot be used in all applications. The reason is that the Verifier has to be able to evaluate $\tilde{f}$ at a random point $r \in \mathbb{F}^v$ to perform the final check in the $\texttt{Sum}-\texttt{check}$ protocol, and in some settings, this computation would be too costly. We will later see problems where we can do this efficiently.

3.3 `FrequencyMoment` Verification

In this part, we will show how to apply $\texttt{Sum}-\texttt{check}$ to a basic streaming problem: `FrequencyMoment` verification. The presentation here is a concrete version of a more generalised protocol by [CTY11].

3.3.1 F_2 Verification

Given an input stream which defines a vector $(f_1, f_2, ..., f_n)$ of length n where $n = 2^b$. Each element in the stream is a pair of values (i, c). Stream length is m. We want to compute $F_2 = \sum_{i=1}^n f_i^2$.
We can think of the frequency vector as a function $P : [n] \to [m]$ such that $P(i) = f_i$. To transform P into a multi-variate function, consider field $\mathbb{F}$ such that $m \subset \mathbb{F}$ and $P : \{0,1\}^b \to \mathbb{F}$. Therefore, $P(i_1, i_2, ...i_b) = f_i$. The MLE $\tilde{P} : \mathbb{F}^b \to \mathbb{F}$ is defined as $\tilde{P}(z_1, z_2...z_b) = \sum_{i \in [n]} P(i_1, i_2, ...i_b)\chi_i(z_1, z_2...z_b)$. Observe that $\forall j \in [n], \tilde{P}(j_1, j_2, ..., j_b) = P(j_1, j_2, ..., j_b) = f_j$. Therefore:
$\sum_{j \in [n]} \tilde{P}(j_1, j_2, ..., j_b)^2 = \sum_{j \in [n]} P(j_1, j_2, ..., j_b)^2 = \sum_{j \in [n]} f_j^2 = F_2$
Thus, we will apply sum-check protocol on $\tilde{P}^2$.
Protocol. Before observing the stream, the Verifier picks a random location $r = (r_1, ..., r_b) \in \mathbb{F}^b$. Then, V

evaluates the MLE $\tilde{P}(r)$ by keeping a running variable Q. For each stream token (i, c), V performs the update $Q \leftarrow Q + c\chi_i(r_1, r_2, ..., r_b)$. Finally, compute $Q^2 = \tilde{P}^2(r)$. From here, V will engage with P similarly to the `Sum − check` protocol with the random point $(r_1, r_2, ..., r_b)$.

Analysis. As `Sum − check` is used for verification, **perfect completeness** is achieved: If Prover is honest, Verifier always accepts. Furthermore, $\tilde{P}^2$ is b-variate with degree at most 2 at all variables. If Prover is dishonest, Verifier rejects with probabilty atlease $1 - 2b/|\mathbb{F}|$. With large enough field, $|\mathbb{F}| = \theta(b/\delta_s) = \theta(1/\delta_s \log n)$, the protocol can have **soundness** δ_s. Communication-wise, there are $O(b) = O(\log n)$ **messages**. On the Verifier's side, they have to maintain space for Q^2, which takes $O(\log|\mathbb{F}|)$ bits. Also, space usage for `Sum − check` is $O(2\log(|\mathbb{F}|))$ bits. Thus, Verifier needs $O(\log|\mathbb{F}|) = O(\frac{1}{\delta_s}\log(n))$ bits.

3.3.2 Extending to F_k

Based on the observation $\forall j \in [n], \tilde{P}(j_1, j_2, ..., j_b) = P(j_1, j_2, ..., j_b) = f_j$, we can apply the same protocol as in F_2, but replace 2 with k:
$\sum_{j\in[n]} \tilde{P}(j_1, j_2, ..., j_b)^k = \sum_{j\in[n]} P(j_1, j_2, ..., j_b)^k = \sum_{j\in[n]} f_j^k = F_k$
Apply `Sum − check` on $\tilde{P}^k$ by computing Q^k from the stream.
All analysis will be the same, but the size of the field $\mathbb{F}$ will blow up by a constant k. Consequently, Verifier will require a space of $O(\frac{k}{\delta_s}\log(n))$ bits. This is an example of the space-advantage of verification over direct computation. Recall that in the AMS sketch, the space complexity is $O(k\epsilon^{-2}n^{1-1/k}\log(1/\delta)\log(mn))$.

3.3.3 Counting distinct elements - F_0

An observation arises from the use of finite fields: If $\mathbb{F}_p$ is defined by the prime p, then by Fermat's Little Theorem, $\forall x \in \mathbb{F}_p, x^{p-1} = 1$. Consequently, $\tilde{P}^{p-1} = F_0$.
It may seem tempting to apply `Sum − check` directly to $\tilde{P}^{p-1}$ to verify F_0. However, this will not work as the $k = p - 1$ is now dependent on the size of the field. The observation $\tilde{P}^{p-1} = F_0$ from [CMT12] is a building block allowing us to take advantage of an important theorem by [GKR15]. We will not be presenting this paper and its result, but the idea is roughly this: If we can model the computation problem as an arithmetic circuit of addition and multiplication over a field $\mathbb{F}$, there is an efficient protocol for verifying it.

4 Verification of Streaming Point-wise problems

In this section, we will discuss a different way of applying MLE in solving a few problems, following the structure of the paper by Chakrabarti et. al [Cha+15]. First, we start by reducing the `Index` problem to `PolynomialEvaluation`, and show that the protocol there can be generalised easily. Later, we will explore a series of problems that can be reduced to `PolynomialEvaluation`.

4.1 `Index` Verification

In the classic `Index` problem, Alice holds a string $x \in \{0, 1\}^n$, and Bob holds an index $j \in [n]$. Bob will have to output the exact value of $x[j]$. To turn it into a SIP problem, we look at the stream $\sigma_x = (x[1], x[2], ..., x[n])$ that Alice sends to Bob. Bob will then process the stream without considering the query j. Finally, using some rounds of communication, Bob will verify that the all-knowing Merlin really has the answer to $x[j]$.
As mentioned in the introduction, we will try to look at this problem through the lens of polynomials. Let $b = \log(n)$ and consider the binary representation of an integer $k = k_1k_2...k_b$, then the operation of finding the $k-$th bit of x is equivalent to evaluating a $b-$variate boolean function P_x at the point $(k_1, k_2, ..., k_b)$. Specifically, $P_x : \{0, 1\}^n \to \{0, 1\}$ and the stream σ_x is an encoding of P_x at all $n = 2^b$ locations. The order of the stream implicitly encoddes the sequence of all input-output pairs: $k-$th element is the value of P_x at the $k-$th point. As

P_x is a boolean function, we can apply MLE and obtain the function $\tilde{P_x} : \mathbb{F}^b \to \mathbb{F}$ such that $\tilde{P_x}(k_1, k_2, ..., k_b) = \sum_{z\in\{0,1\}^b} P_x(z)\chi_z(k_1, k_2, ..., k_b)$ for a large enough field $\mathbb{F}$. We will demonstrate how Bob can verify Merlin's knowledge of $\tilde{P_x}$ at the query point j.

Protocol. Just like in the $\texttt{Sum}-\texttt{check}$ protocol, Bob will compute $\tilde{P_x}$ at a point $r = (r_1, r_2, ..., r_b)$ drawn uniformly random from $\mathbb{F}^b$. To do this, keep a counter Q, and at the $k-$th stream token x_k, update $Q \leftarrow Q + x_k\chi_k(r)$. If the query $j = r$, then Bob does not have to consult Merlin. We will consider the case $j \neq r$. Instead of interacting with Merlin many times like in $\texttt{Sum}-\texttt{check}$, Bob will send a representation of line ℓ through j and r (Theorem 2). Note that there are a finite number of points on this line, meaning that we can assign an order to them (lexicographical order in this case). Furthermore, this line is unique (Theorem 3), so we can describe the line by specifying only the first 2 points of it. To do this, use a degree-1 polynomial $\lambda_\ell : W \to \mathbb{F}^b$ such that $\lambda_\ell(0)$ and $\lambda_\ell(1)$ are these points. For simplicity, we will assume λ_ℓ can be computed efficiently in both runtime and space. In short, Bob will send the function λ_ℓ to Merlin. Next, Merlin will have to send to Bob the polynomial $h : W \to \mathbb{F}$ such that $h = \tilde{P_x} \circ \lambda_\ell$. Finally, Bob knows the points $w, t \in \mathbb{F}$ such that $\lambda_\ell(w) = j, \lambda_\ell(t) = r$, so he will evaluate h at t. If $h(t) = Q$, then Bob trusts Merlin and outputs $x[j] = h(w)$. Otherwise, Bob rejects Merlin. There are **2 messages** sent in total: Bob sending λ_ℓ, and Merlin sending h.

Analysis. If Merlin is honest, then $h(w) = \tilde{P_x}(\lambda_\ell(w)) = \tilde{P_x}(j_1, j_2, ..., j_b) = x[j]$ by definition. The protocol thus has **perfect completeness**. On the other hand, if Merlin is dishonest, then Bob is only tricked if h agrees with $\tilde{P_x} \circ \lambda_\ell$ at t. Note that $h(W) - \tilde{P_x}(\lambda_\ell(W))$ is a polynomial of degree at most b. From Merlin's perspective, r is drawn uniformly random from the line $\ell\backslash\{j\}$, Pr$[h(t) - \tilde{P_x}(\lambda_\ell(t)) = 0] =$ Pr$[h(t) - Q = 0] \leq b/(|\mathbb{F}| - 1)$ (following Theorem 1). By setting $\mathbb{F}$ such that $|\mathbb{F}| = \Theta(b/\delta_s)$ for some small δ_s, then Pr$[h(t) - Q = 0] \leq \delta_s$. Therefore, the protocol has **soundness** δ_s. As for Bob's space usage, keeping $r \in \mathbb{F}^b$ takes $O(b\log(|\mathbb{F}|))$ bits. Bob sends degree-1 λ_ℓ and receives h of degree at most b, so the overall cost is dominated by storing h, which is also $O(b\log(|\mathbb{F}|))$ bits. Overall, Bob needs $O(b\log(|\mathbb{F}|)) = O(b\log(\frac{b}{\delta_s}))$, which is $O(\log(n)(\log(\log(n)/\delta_s)))$. To be more concise, we can write the complexity as $\tilde{O}(\frac{1}{\delta_s}\log(n))$

4.2 Generalised `PolynomialEvaluation` Verification Protocol

A key observation in the `Index` protocol is that the setting of $x \in \{0,1\}^n$ is not very relevant. The whole idea is to reduce the problem to verifying $\tilde{P_x}$ at the desired point. Therefore, we can extend this protocol by ignoring the `Index` setting altogether. The generalised `PolynomialEvaluation` verification protocol becomes: Let the stream σ implicitly describe a b-variate polynomial $P : \mathbb{F}^b \to \mathbb{F}$, e.g. by having the $s-$th token be the value at the $s-$th point according to the lexicographical order. Assume for any point $r \in \mathbb{F}^b$, Bob can compute $P(r)$ by processing the stream (efficiently) using space S. Then Bob can verify Merlin's claim of knowing $P(j)$ for the query j with:

- Perfect completeness.
- Soundness bounded by $b/(|\mathbb{F}| - 1)$, as verification step only utilises the degrees of P and the polynomial sent by Merlin. If possible, by choosing $\mathbb{F}$ carefully, we can achieve any non-perfect soundness set by the parameter $\delta_s \in (0,1)$.
- Space complexity $O(b\log(|\mathbb{F}|) + S)$, as Bob needs $O(S)$ bits to compute $P(r)$ and another $O(b\log(|\mathbb{F}|))$ bits to ineract with Merlin.
- Only 2 messages.

With this, we have a second framework for devising verification protocols: Come up a polynomial P that can be described (implicitly) by the stream, and can be computed efficiently at a random point r. Automatically, we will have a space-efficient 2-message verification protocol like above.

4.3 `FrequencyPointQuery`

We will now see the application of `PolynomialEvaluation` in `FrequencyPointQuery` problem within the context of SIP. Let the stream σ from the universe $[n]$ (WLOG let $n = 2^b$) encode the frequency vector $(f_1, f_2, ..., f_n)$. The problem asks for the retrieval of f_j after the stream has been processed. In the SIP setting, Alice will send to Bob the stream σ, and Bob will verify that Merlin indeed has the answer to f_j. One assumption is added to this problem: There is a known bound to the frequencies, i.e. $|f_i| \le q \forall i \in [n]$.
Using ideas similar to section 3.3.1, we can view the frequency vector as a boolean function $g(i) = f_i$, and apply MLE to get $\tilde{g} : \mathbb{F}^b \to \mathbb{F}$. It is easy to see that $\tilde{g}(j_1, j_2, ..., j_b) = g(j_1, j_2, ..., j_b) = f_j$. Also, we know how to evaluate $\tilde{g}$ at a random point $r = (r_1, r_2, ..., r_b)$ as well, using $O(\log(|\mathbb{F}|))$ bits where $\mathbb{F}$. With these ingredients, we can apply the recipe in section 4.2 to achieve a 2-message protocol that verifies $f_j = \tilde{g}(j_1, j_2, ..., j_b)$. The final space usage of Bob is $O(b \log(|\mathbb{F}|))$.
The priori bound q is needed since $\mathbb{F}$ is finite, and the evaluation of $\tilde{g}$ needs to generate enough distinct values to avoid collisions. Therefore, the characteristic char($\mathbb{F}$) $\ge 2q + 1$. To achieve this and help bound the Soundness to δ_s, we can choose $|\mathbb{F}| = \Theta(\frac{1}{\delta_s}(b + q))$. Thus, Bob actually needs $O(\log(n) \log(\delta_s(q + \log(n)))$ bits.

4.4 `MedianQuery`

Next, a more complicated application of the `PolynomialEvaluation` protocol is the `MedianQuery` problem. We will show how this problem is reduced to a series of problems that are ultimately solved by applying `FrequencyQuery`.
First, we define the problem. Given a data stream σ of length m and a rank $\rho \in [m]$, we want to output j such that: $\sum_{k<j} f_k < \rho$ and $\sum_{k>j} f_k \le m - \rho$. This is also the setting of the `Selection` problem (which we will not go into). For `MedianQuery`, the desired rank is $\rho = \lfloor m/2 \rfloor$.
To solve `MedianQuery`, we can reduce it to the `RangeCount` problem, which utilises `FrequencyQuery`: The `RangeCount` problem takes in an input data stream $\sigma = x_1 x_2 ... x_m$, where given a target range R^* from the set of ranges $\mathcal{R}$, we want to output $|\{j : x_j \in R^*\}|$, which is to maintain a frequency of the number of elements in the target range that appeared in the stream. Note, this is essentially the same as `FrequencyQuery` problem but instead of estimating the frequency of a single point, we are estimating the frequency of elements fall within R^*.
The general idea to verify `RangeCount` is to transform the stream σ to σ' by: for each token $i \in \sigma$, Bob inserts all the ranges R such that $i \in R$ into the stream. After processing σ, Bob will process σ' which has ranges as tokens whose frequencies are f_R. Finally, Bob will verify f_{R^*} with Merlin similarly to `FrequencyQuery`.
Back to `MedianQuery`, we observe that Bob's verification process on the median provided by Merlin is essentially using `RangeCount` to verify that the total frequency of elements from the range $(j, ..., n)$ is $\ge m - \rho + 1$, and the total frequency of elements from the range of $(j + 1, ..., n)$ is $\le m - \rho$. We will not go into great details about the exact protocol, but will try to present some key results:

- There is a 2-message SIP with $O(\log |\mathcal{R}| \log(\log |\mathcal{R}| + m))$ cost for `RangeCount`. The size of the range set $|\mathcal{R}|$ can be very large since it contains a set of ranges of which each range contains m values, so checking for membership might be inefficient.
  However, there exists a lemma that states we can assume that there exists a (poly(S)-time unifrom) de Morgan formula ϕ of length S that takes in the binary input string representing a point x_i and the label of a range $R \in \mathcal{R}$, and outputs a bit that is 1 if an only if $x_i \in R$. With this, we have the next result.
- Suppose membership in ranges R can be decided by de Morgan formulas of length S as above. Then there is a two-round SIP for `RangeCount` problem on `R`, with costs at most $O(S \log(m + S))$, in which the Verifier (Bob) runs in time $O(S)$ per stream update, and the Prover (Merlin) runs in total time $m \cdot \text{poly}(S)$.

By knowing the above theorems for the `RangeCount` problem, the protocol for `MedianQuery` then follows.
Protocol. Bob runs two parallel independent instances of the stream processing step of the `RangeCount` protocol using the class of ranges $\mathcal{R} = \{\{i, i+1, ..., n\} : 1 \le i \le n\}$. In the first round of communication, Merlin sends

the supposed median value $j \in [n]$. The two `RangeCount` instances are now used to check if the frequency of elements from the range $(j, ..., n)$ is $\geq m - \rho + 1$ and the frequency of elements from the range of $(j+1, ..., n))$ is $\leq m - \rho$. This would yield two rounds of verification.

Analysis. Now, note that the size of the range set, $|\mathcal{R}| = n$ and length of the data stream, $|\sigma| = m$, so the space usage follows is $O(\log(n)\log(m + \log(n)))$. Additionally, since we know that a membership in an interval of the form $\{i, ..., n\}$ can be computed by the de Morgan formula of length $O(logn)$, then the claim of Bob running in time poly$(\log n, \log m)$ per update, and Prover (Merlin) runs in total time $m \cdot$ poly$(\log n, \log m)$ will follow, which proves the theorem for `MedianQuery`. As the analysis for completeness and soundness of `MedianQuery` follows the results of `RangeCount`, we will not describe how to achieve that here.

5 Conclusion and Future Directions

As we have explored throughout this report, SIPs represent a crucial aspect of large data stream processing as it addresses the challenges of verifying results from computationally powerful external parties. Throughout this report, we have studied the attributes of SIP protocols and a powerful mathematical tool in the form of Multilinear Extension. MLE provides a way to amplify the characteristics of the original boolean function, which is useful in distinguishing honest and dishonest claims. Two key protocols addressed in this report are `Sum` $-$ `check` and `PolynomialEvaluation`. The former was shown to have applications in efficiently verifying aggregation problems like F_2 and F_k. Meanwhile, we saw that the latter can be used to verify specific point-related problems such as `Index`, `FrequencyQuery`, or `MedianQuery`. These problems have demonstrated the versatility of SIPs in handling a variety of verification tasks.

For future directions, investigation into communication complexity classes within SIP will be important to provide deeper understanding about its efficiency and limitations of these protocols. The paper [Cha+15] goes into great depths about classifying different classes of SIP problems. Another direction would be to bring SIPs into reality by exploring their implementations. A framework was theorised in the 2008 version of [GKR15] and realised by [CMT12]. We find practical applications of SIP to be particularly exciting, as currently, the most popular methods of verification still involve hashing which can be a burden on low-power chips.

References

[Cha+15] Amit Chakrabarti et al. “Verifiable stream computation and Arthur-Merlin communication”. In: *Leibniz international proceedings in informatics (LIPIcs)* (2015), pp. 217–243.

[CTY11] Graham Cormode, Justin Thaler, and Ke Yi. “Verifying computations with streaming interactive proofs”. In: *arXiv preprint arXiv:1109.6882* (2011).

[Tha20a] Justin Thaler. *COSC544 Lecture 2: The power of randomness: Fingerprinting and Freivalds' Algorithm for verifying matrix multiplication. Low-degree and multilinear extensions.* 2020. URL: https://people.cs.georgetown.edu/jthaler/COSC544/Lecture2slides.pdf.

[Tha20b] Justin Thaler. *COSC544 Lecture 3: LFKN's sum-check protocol.* 2020. URL: https://people.cs.georgetown.edu/jthaler/COSC544/Lecture3slides.pdf.

[CMT12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. “Practical verified computation with streaming interactive proofs”. In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. 2012, pp. 90–112.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. “Delegating computation: interactive proofs for muggles”. In: *Journal of the ACM (JACM)* 62.4 (2015), pp. 1–64.