Batch WHIR for Polynomials with Different Sizes

This short write-up summarizes a method for opening multiple multilinear polynomials (MLEs) using the WHIR polynomial commitment scheme [Arn+24]. This write-up assumes the reader is already familiar with the details of general polynomial commitments and the WHIR protocol. If not, you can read this blog for reference.

The basic version of WHIR does not support batch opening commitments to multiple MLEs with different sizes. This feature, however, is often useful when we design SNARKs that involve committing to witness vectors with different sizes. For example, in Spartan [Set20], the prover commits to (and later opens) both vectors of the size of the matrix dimension (the witness vector) and vectors of the size of the number of non-zero entries of the matrix (to commit to the circuit-description matrices themselves). We can save both the prover work and the verifier costs by batching the opening of the MLEs.

Easy case: Batch MLEs with the Same Size at the Same Point

For easier understanding of the full solution, let me describe the simple idea for batch opening commitments to multiple MLEs with the same size at the same point. This idea applies to any code-based polynomial commitment schemes. For simplicity, let me use the example of two polynomials f and g, since the idea can be easily extended to more than two MLEs.

For all existing code-based polynomial commitment schemes, the commitment to the polynomial f is a Merkle root for the Merkle tree built over an encoding of f. This encoding may either be based on the coefficients or the evaluation form of f. Either case, the key property of this encoding is that it is *linear*. As a consequence, given commitments to any two polynomials f and g, we can treat this pair of Merkle roots as if it is a single Merkle root for any linear combination of f and g. For example, to open the i-th entry in the encoding of polynomial $f + g$, we can respectively open the i-th entry in the encoding of f and the i-th entry in the encoding of g, and the verifier adds them together.

By treating the two Merkle roots as if they were a "virtual" Merkle root for the single polynomial $f + g$, we can execute whatever protocol that is originally designed for a single polynomial on this virtual commitment. As a result, the prover can convince the verifier that $f + g$ evaluates to some value y at the point z given by the verifier.

Given the above observation, to prove to the verifier that $f(z) = y_1$ and $g(z) = y_2$ simultaneously, the prover can receive a random challenge α from the verifier, then convince the verifier that $f + \alpha g$ evaluates to $y_1 + \alpha y_2$ at the given point z.

Batch MLEs with the Same Size at Different Points

Next, we consider the case where the prover wants to prove to the verifier that $f(z_1) = y_1$ and $g(z_2) = y_2$ simultaneously, where z_1 and z_2 are two different points. This case is no longer as trivial as the previous case, and we need to exploit the tools specific for MLEs, i.e., from now on let us assume f and g are MLEs with n variables (so their size is $N = 2^n$), and let z_1 and z_2 be vectors of size n, replacing them with vector notations $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$.

For now, we still have a general method for batching that works for all code-based polynomial commitment schemes (although only works for MLEs). The idea is to use sum-check to transform the evaluation points into a single one. The verifier samples the random α and sends to the prover. The parties engage in the sum-check protocol for the following statement:

$$y_1 + \alpha \cdot y_2 = \sum_{\boldsymbol{b} \in \{0,1\}^n} f(\boldsymbol{b}) \cdot eq(\boldsymbol{b}, \boldsymbol{z}_1) + \alpha \cdot g(\boldsymbol{b}) \cdot eq(\boldsymbol{b}, \boldsymbol{z}_2)$$

where $eq(\boldsymbol{b}, \boldsymbol{X})$ is the Lagrange basis polynomial over the hypercube for the point $\boldsymbol{b}$.

At the end of the sum-check protocol, the prover remains to prove to the verifier that $f(\boldsymbol{r}) = y_1'$ and $g(\boldsymbol{r}) = y_2'$ for some random $\boldsymbol{r}$ generated during the protocol. This is exactly the easy case in the previous section.

Batch MLEs with Different Sizes at Different Points

Now we are ready to deal with the case where f and g may have different number of variables. In this case, we need to exploit more details of the WHIR protocol, since there is no technique that works for all code-based polynomial commitment schemes simultaneously.

Recall that WHIR is essentially a protocol for reducing an inner-product statement into another one with a smaller size. In more detail, given the commitment to f and a publicly known weight polynomial w (which has a succinct description and the verifier can evaluate this polynomial efficiently) and the target sum σ, after executing one round of WHIR, the parties reach a new statement: a commitment to a new polynomial f' and a new weight polynomial w' (which is also publicly known and efficiently evaluatable by the verifier) with a new target sum σ'. In WHIR, f' is related to f by substituting k variables of f by random challenges, that means f' has $n-k$ variables. However, the codeword size in the commitment is half of the original codeword size for f, i.e., the log of the codeword size is reduced by only 1 instead of k.

Two key facts to note are:

1. The encoding is coefficient-based. In more detail, to commit to the MLE f, let the vector $\boldsymbol{f}$ be the coefficients of f in little-endian, then treat $\boldsymbol{f}$ as the coefficient vector of a univariable polynomial $\hat{f}$. The codeword is produced by evaluating $\hat{f}$ over a univariate domain $\mathcal{D} \subset \mathbb{F}$. It is easy to see that we can zero-pad the MLE f without affecting the codeword. In another word, encoding a polynomial with $n-k$ variables is the same as encoding it as if it is an n

-variate polynomial with zero coefficients for terms involving the extra variables. The key consequence of this fact is that: *we can directly add (or linearly combine) codewords for MLEs with different number of variables, and the result is the codeword of the added polynomials, as long as the codeword is over the same domain $\mathcal{D}$.*

2. Although the polynomial size is reduced by 2^k in each WHIR round, the codeword size is reduced by only a half. As a result, by repeatedly applying the WHIR round to f, the size of the codeword of f will eventually be the same as the size of the codeword of g, (here, w.l.o.g. assume f has larger size than g). Although the polynomial sizes do not match, this does not prevent us from linearly combining them in the codeword space, due to the previous fact.

We are now ready to describe the protocol for the simple case with only two MLEs f and g, with n_1 and n_2 variables, respectively. The prover is trying to convince the verifier that $f(\boldsymbol{z}_1) = y_1$ and $g(\boldsymbol{z}_2) = y_2$, where $\boldsymbol{z}_1$ is a vector of length n_1 and $\boldsymbol{z}_2$ is a vector of length n_2. Assume $n_1 > n_2$.

First, let $w_1(\boldsymbol{X}) = eq(\boldsymbol{z}_1, \boldsymbol{X})$. The parties run the WHIR protocol for $n_1 - n_2$ rounds for f and the weight function w_1 and target sum $\sigma = y_1$, and produce a commitment to f' and a new weight polynomial w' with target sum σ'.

Next, the verifier samples α, and the parties run the sum-check protocol for the statement:

$$y' + \alpha \cdot y_2 = \sum_{\boldsymbol{b} \in \{0,1\}^{n_2}} f'(\boldsymbol{b}) \cdot w'(\boldsymbol{b}) + \alpha \cdot g(\boldsymbol{b}) \cdot eq(\boldsymbol{b}, \boldsymbol{z}_2).$$

At the end of this sum-check, we are back to the easy case: batch opening f' and g with the same number of variables n_2 at the same point.

The above protocol can be easily generalized to more MLEs with more sizes:

1. Pick up all the MLEs with the maximal codeword size.
2. Run the sum-check to unify the evaluation point (this can be skipped if there is only one MLE for this size).
3. Run the WHIR protocol for a random linear combination of these MLEs, until we obtain a codeword with the same size as the codeword for the second-largest MLE in the batch.
4. Repeat the above steps until all the codewords are merged into one, then run the WHIR protocol as usual until the end.

References

Arn+24. Arnon, G., Chiesa, A., Fenzi, G., Yogev, E.: WHIR: Reed–Solomon Proximity Testing with Super-Fast Verification, (2024)

Set20. Setty, S.: Spartan: Efficient and General-Purpose zkSNARKs without Trusted Setup. In: Annual International Cryptology Conference. pp. 704–737. Springer (2020)