COSC 544 Probabilistic Proof Systems 9/19/17

An Optimal Interactive Proof for Matrix Multiplication

Lecturer: Justin Thaler

1 A Second Application of Sum-Check: An Optimal Interactive Proof for Matrix Multiplication

This section describes a highly optimized IP protocol for matrix multiplication (MATMULT) from [Tha13]. While this MATMULT protocol is of interest in its own right, it is included here for didactic reasons: it displays, in a clean and unencumbered setting, all of the algorithmic insights that are exploited later in this survey to give more general IP and MIP protocols.

Given two $n \times n$ input matrix A, B over field $\mathbb{F}$, the goal of MATMULT is to compute the product matrix $C = A \cdot B$.

1.1 Background: Freivalds' Protocol.

Recall from Lecture 2 that, in 1979, Freivalds [Fre79] gave the following verification protocol for MATMULT: to check that $A \cdot B = C$, $\mathcal{V}$ picks a random vector $\mathbf{x} \in \mathbb{F}^n$, and accepts if $A \cdot (B\mathbf{x}) = C\mathbf{x}$. $\mathcal{V}$ can compute $A \cdot (B\mathbf{x})$ with two matrix-vector multiplications, which requires just $O(n^2)$ time. Thus, in Freivelds' protocol, $\mathcal{P}$ simply finds and sends the correct answer C, while $\mathcal{V}$ runs in optimal $O(n^2)$ total time. Today, Freivalds' protocol is regularly covered in introductory textbooks on randomized algorithms.

At first glance, Freivalds' protocol seems to close the book on verification protocols for MATMULT, since the runtimes of both $\mathcal{V}$ and $\mathcal{P}$ are optimal: $\mathcal{P}$ does *no* extra work to prove correctness of the answer matrix C, $\mathcal{V}$ runs in time linear in the input size, and the protocol is even non-interactive ($\mathcal{P}$ just sends the answer matrix C to $\mathcal{V}$).

However, there is a sense in which it is possible to improve on Freivalds' protocol by introducing interaction between $\mathcal{P}$ and $\mathcal{V}$. In many settings, algorithms invoke MATMULT, but they are not really interested in the full answer matrix. Rather, they apply a simple post-processing step to the answer matrix to arrive at the quantity of true interest. For example, the best-known graph diameter algorithms repeatedly square the adjacency matrix of the graph, but ultimately they are not interested in the matrix powers – they are only interested in a single number. As another example, standard algorithms for triangle-counting invoke matrix multiplication, but are also interested in just a single number.

If Freivalds' protocol is used to verify the matrix multiplication steps of these algorithms, the actual product matrices must be sent for each step, necessitating $\Omega(n^2)$ communication. In practice, this can easily be many terabytes of data, even on graphs G with a few million nodes (also, even if G is sparse, powers of G's adjacency matrix may be dense).

This section describes an interactive matrix multiplication protocol from [Tha13] that preserves the runtimes of $\mathcal{V}$ and $\mathcal{P}$ from Freivalds' protocol, but avoids the need for $\mathcal{P}$ to send the full answer matrix in the settings described above—in these settings, the communication cost of the interactive protocol is just $O(\log n)$ field elements per matrix multiplication.

Preview: The Power of Interaction. This comparison of the interactive MATMULT protocol to Freivalds' non-interactive one exemplifies the power of interaction in verification. Interaction buys the verifier the ability to ensure that the prover correctly materialized intermediate values in a computation (in this case, the entries of the product matrix C), without requiring the prover to explicitly materialize those values to the verifier. $\square$

Preview: Other Protocols for MATMULT. An alternate interactive MATMULT protocol can be obtained by applying the GKR protocol (covered in the next lecture) to a circuit $\mathcal{C}$ that computes the product C of two input matrices A, B. The verifier in this protocol runs in $O(n^2)$ time, and the prover runs in time $O(S)$, where S is the number of gates in $\mathcal{C}$.

The advantage of the MATMULT protocol described above is two-fold. First, the protocol of this section does not care how the prover finds the right answer. In contrast, the GKR protocol demands that the prover compute the answer matrix C in a prescribed manner, namely by evaluating the circuit $\mathcal{C}$ gate-by-gate. Second, the prover in the protocol of this section simply finds the right answer and then does $O(n^2)$ extra work to prove correctness. This $O(n^2)$ term is a low-order additive overhead, assuming that there is no linear-time algorithm for matrix multiplication. In contrast, the GKR protocol introduces a constant factor overhead for the prover. In practice, this is the difference between a prover that runs several times slower than an (unverifiable) MATMULT algorithm, and a prover that runs a fraction of a percent slower [Tha13]. $\square$

2 The Protocol

Given $n \times n$ input matrix A, B, recall that we denote the product matrix $A \cdot B$ by C. Interpret A, B, and C as functions mapping $\{0,1\}^{\log n} \times \{0,1\}^{\log n}$ to $\mathbb{F}$ via:

$$A(i_1, \dots, i_{\log n}, j_1, \dots, j_{\log n}) = A_{ij}.$$

As usual, $\widetilde{A}$, $\widetilde{B}$, and $\widetilde{C}$ denote the MLEs of A, B, and C.

It is cleanest to describe the protocol for MATMULT as a protocol for evaluating $\widetilde{C}$ at any given point $(\mathbf{r}_1, \mathbf{r}_2) \in \mathbb{F}^{\log n \times \log n}$. As we explain later (cf. Section 3.1), this turns out to be sufficient for application problems such as graph diameter and triangle counting.

The protocol for computing $\widetilde{C}(\mathbf{r}_1, \mathbf{r}_2)$ exploits the following explicit representation of the polynomial $\widetilde{C}(\mathbf{x}, \mathbf{y})$.

Lemma 2.1. $\widetilde{C}(\mathbf{x}, \mathbf{y}) = \sum_{\mathbf{b} \in \{0,1\}^{\log n}} \widetilde{A}(\mathbf{x}, \mathbf{b}) \cdot \widetilde{B}(\mathbf{b}, \mathbf{y})$. *Here, the equality holds as formal polynomials in the coordinates of* $\mathbf{x}$ *and* $\mathbf{y}$.

Proof. The left and right hand sides of Equation (2.1) are both multilinear polynomials in the coordinates of $\mathbf{x}$ and $\mathbf{y}$. Since the MLE of C is unique, we need only check that the left and right hand sides of Equation (2.1) agree for all *Boolean* vectors $\mathbf{i}, \mathbf{j} \in \{0,1\}^{\log n}$. That is, we must check that for Boolean vectors $\mathbf{i}, \mathbf{j} \in \{0,1\}^{\log n}$,

$$C(\mathbf{i}, \mathbf{j}) = \sum_{\mathbf{k} \in \{0,1\}^{\log n}} A(\mathbf{i}, \mathbf{k}) \cdot B(\mathbf{k}, \mathbf{j}). \tag{1}$$

But this is immediate from the definition of matrix multiplication. $\square$

With Lemma 2.1 in hand, the interactive protocol is immediate: we compute $\widetilde{C}(\mathbf{r}_1, \mathbf{r}_2)$ by applying the sum-check protocol to the $(\log n)$-variate polynomial $g(\mathbf{z}) := \widetilde{A}(\mathbf{r}_1, \mathbf{z}) \cdot \widetilde{B}(\mathbf{z}, \mathbf{r}_2)$.

Example. When trying to understand Lemma 2.1, it is illustrative to consider an explicit example. Consider the 2×2 matrices $A = \begin{bmatrix} 0 & 1 \\ 2 & 0 \end{bmatrix}$ and $B = \begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix}$ over $\mathbb{F}_5$. One can check that $A \cdot B = \begin{bmatrix} 0 & 4 \\ 2 & 0 \end{bmatrix}$

Viewing A and B as a functions mapping $\{0,1\}^2 \to \mathbb{F}_7$,

$$\widetilde{A}(x_1,x_2) = (1-x_1)x_2 + 2x_1(1-x_2) = -3x_1x_2 + 2x_1 + x_2,$$

and

$$\widetilde{B}(x_1,x_2) = (1-x_1)(1-x_2) + 4x_1x_2 = 5x_1x_2 - x_1 - x_2 + 1 = 1 - x_1 - x_2,$$

where the final equality used the fact that we are working over $\mathbb{F}_5$, so the coefficient 5 is the same as the coefficient 0.

Observe that

$$\begin{aligned} \sum_{b \in \{0,1\}} \widetilde{A}(x_1,b) \cdot \widetilde{B}(b,x_2) \quad &= \widetilde{A}(x_1,0) \cdot \widetilde{B}(0,x_2) + \widetilde{A}(x_1,1) \cdot \widetilde{B}(1,x_2) \\ &= 2x_1 \cdot (1-x_2) + (-x_1+1) \cdot (-x_2) = -x_1x_2 + 2x_1 - x_2. \end{aligned} \tag{2}$$

Meanwhile, viewing C as a function mapping $\{0,1\}^2 \to \mathbb{F}_5$, we can calculate via Lagrange Interpolation:

$$\widetilde{C}(x_1,x_2) = 4(1-x_1)x_2 + 2x_1(1-x_2) = -6x_1x_2 + 2x_1 + 4x_2 = -x_1x_2 + 2x_1 - x_2,$$

where the final equality uses that $6 \cong 1$ and $4 \cong -1$ when working modulo 5. Hence, we have verified that Lemma 2.1 indeed holds for this particular example.

3 Discussion of costs.

Rounds and communication cost. Since g is a $(\log n)$-variate polynomial of degree 2 in each variable, the total communication is $O(\log n)$ field elements, spread over $\log n$ rounds.

$\mathcal{V}$'s runtime. At end of sum-check, $\mathcal{V}$ must evaluate $g(\mathbf{r}_3) = \widetilde{A}(\mathbf{r}_1,\mathbf{r}_3) \cdot \widetilde{B}(\mathbf{r}_3,\mathbf{r}_2)$. To perform this evaluation, it suffices for $\mathcal{V}$ to evaluate $\widetilde{A}(\mathbf{r}_1,\mathbf{r}_3)$ and $\widetilde{B}(\mathbf{r}_3,\mathbf{r}_2)$. Since $\mathcal{V}$ is given the matrices A and B as input, Lemma 1.8 of Lecture 4 implies that both evaluations can be performed in $O(n^2)$ time.

$\mathcal{P}$'s runtime. Recall that in each round k of the sum-check protocol $\mathcal{P}$ sends a quadratic polynomial $g_i(X_k)$ claimed to equal:

$$\sum_{b_{k+1} \in \{0,1\}} \dots, \sum_{b_{\log n} \in \{0,1\}} g(r_{3,1}, \dots, r_{3,k-1}, X_i, b_{k+1}, \dots b_{\log n}),$$

and to specify $g_k(X_k)$, $\mathcal{P}$ can just send the values $g_i(0), g_i(1)$, and $g_i(2)$. Thus, it is enough for $\mathcal{P}$ to evaluate g at all points of the form

$$\left(r_{3,1}, \dots, r_{3,k-1}, \{0,1,2\}, b_{k+1}, \dots, b_{\log n}\right) : \left(b_{k+1}, \dots, b_{\log n}\right) \in \{0,1\}^{\log n - k}. \tag{3}$$

There are $3 \cdot n/2^k$ such points in round k.

We describe three separate methods to perform these evaluations. The first method is the least sophisticated and requires $\Theta(n^3)$ total time. The second method reduces the runtime to $\Theta(n^2)$ per round, for a

Communication	Rounds	$\mathcal{V}$ time	$\mathcal{P}$ time
$O(\log n)$ field elements	$\log n$	$O\left(n^2\right)$	$T+O(n^2)$

Table 1: Costs of the MATMULT protocol of Section 1 when applied to $n \times n$ matrices A and B. Here, T is the time required by $\mathcal{P}$ to compute the product matrix $C = A \cdot B$.

total runtime bound of $\Theta(n^2 \log n)$ over all $\log n$ rounds. The third method is more sophisticated still – it enables the prover to *reuse work* across rounds, ensuring that $\mathcal{P}$'s runtime in round k is bounded by $O(n^2/2^k)$. Hence, the prover's total runtime is $O(\sum_k n^2/2^k) = O(n^2)$.

Method 1. As described when bounding $\mathcal{V}$'s runtime, g can be evaluated at any point in $O(n^2)$ time. Since there are $3 \cdot n/2^k$ points at which $\mathcal{P}$ must evaluate g in round k, this leads to a total runtime for $\mathcal{P}$ of $O(\sum_k n^3/2^k) = O(n^3)$.

Method 2. To improve on the $O(n^3)$ runtime of Method 1, the key is to exploit the fact that $3 \cdot n/2^k$ points at which $\mathcal{P}$ needs to g in round k are not arbitrary points in $\mathbb{F}^{\log n}$, but are instead highly structured. Specifically, each such point $\mathbf{z}$ is in the form of Equation (3), and hence the trailing coordinates of $\mathbf{z}$ are all Boolean. As explained below, this property ensures that ***each entry A_{ij} of A contributes to $g\left(r_{3,1},\ldots,r_{3,k-1},\{0,1,2\},b_{k+1},\ldots,b_{\log n}\right)$ for only one tuple $(b_{k+1},\ldots,b_{\log n}) \in \{0,1\}^{\log n-k}$, and similarly for each entry of B_{ij}.*** Hence, $\mathcal{P}$ can make a single pass over the matrices A and B, and for each entry A_{ij} or B_{ij}, $\mathcal{P}$ only needs to update $g(\mathbf{z})$ for the three tuples $\mathbf{z}$ of the form $\left(r_{3,1},\ldots,r_{3,k-1},\{0,1,2\},b_{k+1},\ldots,b_{\log n}\right)$.

In more detail, in order to evaluate g at any input $\mathbf{z}$, it suffices for $\mathcal{P}$ to evaluate $\widetilde{A}(\mathbf{r}_1,\mathbf{z})$ and $\widetilde{B}(\mathbf{z},\mathbf{r}_2)$. We'll explain the case of evaluating $\widetilde{A}(\mathbf{r}_1,\mathbf{z})$ at all relevant points $\mathbf{z}$, since the case of $\widetilde{B}(\mathbf{z},\mathbf{r}_2)$ is identical. From Lemma 1.6 of Lecture 4 (Lagrange Interpolation), recall that $\widetilde{A}(\mathbf{r}_1,\mathbf{z}) = \sum_{\mathbf{i},\mathbf{j}\in\{0,1\}^{\log n}} A_{\mathbf{ij}}\chi_{(\mathbf{i},\mathbf{j})}(\mathbf{r}_1,\mathbf{z})$. For any input $\mathbf{z}$ of the form $\left(r_{3,1},\ldots,r_{3,k-1},\{0,1,2\},b_{k+1},\ldots,b_{\log n}\right)$, notice that $\chi_{\mathbf{i},\mathbf{j}}(\mathbf{r}_1,\mathbf{z}) = 0$ unless $(j_{k+1},\ldots,j_{\log n}) = (b_{k+1},\ldots,b_{\log n})$, since for any coordinate ℓ such that $j_\ell \neq b_\ell$, the factor $(j_\ell b_\ell + (1-j_\ell)(1-b_\ell))$ appearing in the product defining $\chi_{(\mathbf{i},\mathbf{j})}$ equals 0 (cf. Equation 2 of Lecture 4).

This enables $\mathcal{P}$ to evaluate $\widetilde{A}(\mathbf{r}_1,\mathbf{z})$ in round k at all points $\mathbf{z}$ of the form of Equation (3) with a single pass over A: when $\mathcal{P}$ encounters entry $A_{\mathbf{ij}}$ of A, $\mathcal{P}$ updates $\widetilde{A}(\mathbf{z}) \leftarrow \widetilde{A}(\mathbf{z}) + \chi_{\mathbf{i},\mathbf{j}}(\mathbf{z})$ for the three relevant values of $\mathbf{z}$.

Method 3. To shave the last factor of $\log n$ off $\mathcal{P}$'s runtime, the idea is to have $\mathcal{P}$ reuse work across rounds. Specifically, if two entries $(\mathbf{i},\mathbf{j}),(\mathbf{i}',\mathbf{j}') \in \{0,1\}^{\log n} \times \{0,1\}^{\log n}$ agree in their last ℓ bits, then $A_{\mathbf{i},\mathbf{j}}$ and $A_{\mathbf{i}',\mathbf{j}'}$ contribute to the same three points in each of the final ℓ rounds of the protocol. The specific points that they contribute to in each round $k \geq \log(n) - \ell$ are the ones of the form $\mathbf{z} = \left(r_{3,1},\ldots,r_{3,k-1},\{0,1,2\},b_{k+1},\ldots,b_{\log n}\right)$, where $b_{k+1}\ldots b_{\log n}$ equal the trailing bis of $(\mathbf{i},\mathbf{j})$ and $(\mathbf{i}',\mathbf{j}')$.

To elide many details, this observation ensures that $\mathcal{P}$ can treat $(\mathbf{i},\mathbf{j})$ and $(\mathbf{i}',\mathbf{j}')$ as a single entity thereafter. There are only $O(n^2/2^k)$ entities of interest after k variables have been bound (out of the $2\log n$ variables over which $\widetilde{A}$ is defined). So the total work that $\mathcal{P}$ invests over the course of the protocol is $O\left(\sum_{k=1}^{2\log n} n^2/2^k\right) = O(n^2)$.

3.1 Why Does Computing $\widetilde{C}(\mathbf{r}_1,\mathbf{r}_2)$ Suffice?

Algorithms often invoke MATMULT to compute some product matrix C, and then apply some post-processing to C to compute an answer that is much "smaller" than C itself (often the answer is just a single number). In these settings, $\mathcal{V}$ can apply a general-purpose protocol, such as the GKR protocol described in Lecture

2, to verify that the post-processing step was correctly applied to the product matrix C. As we will see in Lecture 2, at the end of the application of the GKR protocol, $\mathcal{V}$ needs to evaluate $\widetilde{C}(\mathbf{r}_1, \mathbf{r}_2)$ at a randomly chosen point $(\mathbf{r}_1, \mathbf{r}_2) \in \mathbb{F}^{\log n \times \log n}$. $\mathcal{V}$ can do this using the MATMULT protocol described above.

Crucially, this post-processing step typically requires time linear in the size of C. So $\mathcal{P}$'s runtime in this application of the GKR protocol will be proportional to the size of (a circuit computing) the post-processing step, which is typically just $\tilde{O}(n^2)$. This will be a low-order cost for $\mathcal{P}$, since computing C alone will require time at least n^{ω}, where ω is the matrix multiplication constant (it is widely believed that ω is strictly greater than 2).

As a concrete example, consider the problem of computing the diameter of a directed graph G. Let A denote the adjacency matrix of G, and let I denote the $n \times n$ identity matrix. Then the diameter of G is the least positive number d such that $(A+I)^d_{ij} \neq 0$ for all (i, j). This yields the following natural protocol for diameter. $\mathcal{P}$ sends the claimed output d to V, as well as an (i, j) such that $(A+I)^{d-1}_{ij} = 0$. To confirm that d is the diameter of G, it suffices for $\mathcal{V}$ to check two things: first, that all entries of $(A+I)^d$ are non-zero, and second that $(A+I)^{d-1}_{ij}$ is indeed non-zero.

The first task is accomplished by combining the MATMULT protocol with the GKR protocol as follows. Let d_j denote the jth bit in the binary representation of d. Then $(A+I)^d = \prod_j^{\lceil \log d \rceil} (A+I)^{d_j 2^j}$, so computing the number of non-zero entries of $D_1 = (A+I)^d$ can be computed via a sequence of $O(\log d)$ matrix multiplications, followed by a post-processing step that computes the number of non-zero entries of D_1. We can apply the GKR protocol to verify this post-processing step, but at the end of the protocol, $\mathcal{V}$ needs to evaluate $\widetilde{D}_1(\mathbf{r}_1, \mathbf{r}_2)$. $\mathcal{V}$ cannot do this without help, so $\mathcal{V}$ outsources even this computation to $\mathcal{P}$, by using $O(\log d)$ invocations of the MATMULT protocol described above.

The second task, of verifying that $(A+I)^{d-1}_{ij} = 0$, is similarly accomplished using $O(\log d)$ invocations of the MATMULT protocol – since $\mathcal{V}$ is only interested in one entry of $(A+I)^{d-1}$, $\mathcal{P}$ need not send the matrix $(A+I)^{d-1}$ in full, and the total communication here is just polylog(n).

Ultimately, $\mathcal{V}$'s runtime in this diameter protocol is $O(m \log n)$, where m is the number of edges in G. $\mathcal{P}$'s runtime in the above diameter protocol matches the best known unverifiable diameter algorithm up to a low-order additive term [Sei95, Yus10], and the communication is just polylog(n).

References

[Fre79] Rusins Freivalds. Fast probabilistic algorithms. In *MFCS*, pages 57–69, 1979.

[Sei95] Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. Syst. Sci.*, 51(3):400–403, December 1995.

[Tha13] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *Proceedings of the 33rd Annual Conference on Advances in Cryptology*, CRYPTO'13, Berlin, Heidelberg, 2013. Springer-Verlag.

[Yus10] Raphael Yuster. Computing the diameter polynomially faster than APSP. *arXiv preprint arXiv:1011.6181*, 2010.