Computational Number Theory and Algebra April 25, 2012

Lecture 3

Lecturers: Markus Bläser, Chandan Saha *Scribe: Chandan Saha*

Towards the end of the previous lecture, we mentioned that locally decodable codes (LDC) are very useful in theory as well as in practice. We saw an example of LDC in the form of Hadamard codes. But the issue with Hadamard codes is that the codeword length is a bit too long. We will see a better construction of LDC today, namely Reed-Muller codes. Today's topics of discussions are:

- Reed-Muller codes,
- Chinese Remaindering Theorem (CRT), and
- An application of CRT - determinant computation.

1 Reed-Muller codes

Reed-Muller codes are used in communications. They are also used to design *probabilistically checkable proofs (PCPs)* in computational complexity theory. Instead of univariate polynomials (as is the case for Reed-Solomon codes), Reed-Muller codes use multivariate polynomials to encode the message. The code uses three parameters: a prime power (alphabet size) q, a number of variables n, and degree $d < q - 1$. It encodes a $k = \binom{n+d}{d}$-long q-ary message to a q^n-long codeword. The encoding procedure is as follows.

Algorithm 1 Reed-Muller codes: Encoding

1. Let $m = (m_1, \ldots, m_k) \in \mathbb{F}_q^k$ be the message to be encoded.
2. Let $\mathbf{w}_1, \ldots, \mathbf{w}_k$ be a 'certain' fixed collection of points in $\mathbb{F}_q^n$.
3. Construct a polynomial $P_m(x_1, \ldots, x_n)$ with $\deg(P_m) \leq d$ such that $P_m(\mathbf{w}_i) = m_i$.
4. The evaluations of $P_m(x_1, \ldots, x_n)$ at all points in $\mathbb{F}_q^n$ is the q^n-long codeword.

Step 3 involves solving a system of linear equations by treating the $\binom{n+d}{d}$ coefficients of $P_m(\mathbf{x})$ as variables. (Convince yourself that P_m has at most $\binom{n+d}{d}$ monomials.) Also, there exist points $\mathbf{w}_1, \ldots, \mathbf{w}_k$ such that the corresponding linear system has full rank (why?). So, a solution to the system $P_m(\mathbf{w}_i) = m_i$ $(1 \leq i \leq k)$ can always be found in step 3. Notice that the codeword length is q^n, so that the alphabet size q can be exponentially smaller than the codeword length (unlike Reed-Solomon codes).

The decoding procedure for Reed-Muller codes uses the power of randomization - an indispensible tool in computation (especially in reducing *query complexity*). Let D be the received word. By querying the received word at point $\mathbf{u} \in \mathbb{F}_q^n$, we get a (potentially corrupted) value of $P_m(\mathbf{u})$. The decoding procedure that retrieves the part of the message $m_i = P_m(\mathbf{w}_i)$ is as follows.

Algorithm 2 Reed-Muller codes: Decoding

1. Pick a point $\mathbf{v} \in \mathbb{F}_q^n$ uniformly at random.
2. Query D at points $\{\mathbf{w}_i + y\mathbf{v}\}_{y \in S}$, where $S \subset \mathbb{F}_q^\times$, $|S| = d + 1$, to obtain values $\{d_y\}_{y \in S}$.
3. Find the unique univariate polynomial f, $\deg(f) \leq d$, such that $f(y) = d_y$.
4. Output $f(0)$.

The set S chosen in step 2 is an arbitrary subset of $\mathbb{F}_q^\times$ of size $d + 1$. Notice that as $\mathbf{v}$ is chosen uniformly randomly, $\mathbf{w}_i + y\mathbf{v}$ is also a random point in $\mathbb{F}_q^n$, for any fixed $y \in \mathbb{F}_q^\times$. So, if at most δ fraction of the q^n-long

received word is corrupted then querying D at $\mathbf{w}_i + y\mathbf{v}$ (for any fixed $y \in S$) returns the value $P_m(\mathbf{w}_i + y\mathbf{v})$ with probability at least $1 - \delta$. Therefore, by union bound on the probabilities, we can recover the correct values of $P_m(\mathbf{w}_i + y\mathbf{v})$, for all $y \in S$, with probability at least $1 - (d+1) \cdot \delta$. Now, for a moment think of the univariate polynomial $P_m(\mathbf{w}_i + y\mathbf{v})$ by treating y as a variable. This polynomial has degree at most d, as $\deg(P_m(\mathbf{x})) \leq d$. Also, we know that we can obtain $d+1$ correct evaluations of this polynomial $P_m(\mathbf{w}_i + y\mathbf{v})$ at all $y \in S$ with probability at least $1 - (d+1) \cdot \delta$. Which mean, with probability at least $1 - (d+1) \cdot \delta$, the polynomial $f(y)$ constructed in step 3 is actually $P_m(\mathbf{w}_i + y\mathbf{v})$. If so, then $f(0) = P_m(\mathbf{w}_i) = m_i$. We can find the polynomial $f(y)$ in step 3 using polynomial interpolation.

Thus, Reed-Muller code is a $(d+1, \delta, (d+1) \cdot \delta)$-locally decodable code - meaning that, by making $d+1$ queries to the received word, the decoder finds the right value of m_i with probability at least $1 - (d+1) \cdot \delta$, where δ is an upper bound on the fraction of errors that can occur in the codeword. The chance that the decoder makes an error, can be made independent of d (in the above calculation, it is $(d+1) \cdot \delta$), by using the Berlekamp-Welch algorithm for Reed-Solomon codes - I leave this as an exercise.

2 Chinese Remaindering Theorem

This theorem is a structural result about rings which is used for speeding up computation over integers and polynomials, and also for arguing over rings like in the analysis of the Miller-Rabin primality test (which will be covered in a later lecture). We state the theorem in a general form and then apply it to the rings of integers and polynomials.

Let us refresh the definition of an *ideal* of a ring: A nonempty subset $\mathcal{I}$ of a (commutative) ring $\mathcal{R}$ is an ideal of $\mathcal{R}$ if, $\mathcal{I}$ is a subgroup of $\mathcal{R}$ under addition, and for every $u \in \mathcal{I}$ and $r \in \mathcal{R}$, $u \cdot r$ belongs to $\mathcal{R}$. Two ideals $\mathcal{I}$ and $\mathcal{J}$ of a ring $\mathcal{R}$ are *coprime* if there are elements $a \in \mathcal{I}$ and $b \in \mathcal{J}$ such that $a + b = 1$. The product of two ideals $\mathcal{I}$ and $\mathcal{J}$, denoted by $\mathcal{I}\mathcal{J}$, is the ideal generated by all elements of the form $a \cdot b$ where $a \in \mathcal{I}$ and $b \in \mathcal{J}$. The theorem states the following.

Theorem 1 (Chinese Remaindering Theorem) *Let $\mathcal{I}_1, \ldots, \mathcal{I}_r$ be pairwise coprime ideals of $\mathcal{R}$ and $\mathcal{I} = \mathcal{I}_1 \ldots \mathcal{I}_r$ be their product. Then,*

$$\frac{\mathcal{R}}{\mathcal{I}} \cong \frac{\mathcal{R}}{\mathcal{I}_1} \oplus \ldots \oplus \frac{\mathcal{R}}{\mathcal{I}_r}$$

Moreover, this isomorphism map is given by, $a \bmod \mathcal{I} \longrightarrow (a \bmod \mathcal{I}_1, \ldots, a \bmod \mathcal{I}_r)$, for all $a \in \mathcal{R}$.

Proof The proof uses induction on the number of coprime ideals. Let $\mathcal{J} = \mathcal{I}_2 \ldots \mathcal{I}_r$. Since $\mathcal{I}_1$ is coprime to $\mathcal{I}_j$ for every j, $2 \leq j \leq r$, there are elements $y_j \in \mathcal{I}_j$ and $x_j \in \mathcal{I}_1$ such that $x_j + y_j = 1$. Therefore, $\prod_{j=2}^{r} (x_j + y_j) = x + y' = 1$ where $x \in \mathcal{I}_1$ and $y' \in \mathcal{J}$, implying that $\mathcal{I}$ and $\mathcal{J}$ are coprime.

We claim that $\mathcal{I} = \mathcal{I}_1 \cap \mathcal{J}$. By definition, $\mathcal{I} = \mathcal{I}_1\mathcal{J}$ and it is easy to see that $\mathcal{I}_1\mathcal{J} \subseteq \mathcal{I}_1 \cap \mathcal{J}$. If $z \in \mathcal{I}_1 \cap \mathcal{J}$ then, from $x + y' = 1$ we have $zx + zy' = z$. The left hand side of the last equation is an element of $\mathcal{I}_1\mathcal{J}$ as both $zx, zy' \in \mathcal{I}_1\mathcal{J}$. Therefore, $\mathcal{I}_1 \cap \mathcal{J} = \mathcal{I}_1\mathcal{J} = \mathcal{I}$.

Consider the map $\phi : \frac{\mathcal{R}}{\mathcal{I}} \to \frac{\mathcal{R}}{\mathcal{I}_1} \oplus \frac{\mathcal{R}}{\mathcal{J}}$ defined as $\phi(a \bmod \mathcal{I}) = (a \bmod \mathcal{I}_1, a \bmod \mathcal{J})$. It is easy to check that ϕ is well-defined and is in fact a homomorphism. Let $a_1 = a \bmod \mathcal{I}_1$ and $a' = a \bmod \mathcal{J}$. We will abuse notation slightly and write $\phi(a) = (a_1, a')$.

If $\phi(a) = \phi(b) = (a_1, a')$ then $a_1 = a \bmod \mathcal{I}_1 = b \bmod \mathcal{I}_1$, implying that $a - b \in \mathcal{I}_1$. Similarly, $a - b \in \mathcal{J}$. This means $a - b \in \mathcal{I} \cap \mathcal{J} = \mathcal{I}$ and hence ϕ is a one-one map. Also, since $x + y' = 1$ for $x \in \mathcal{I}_1$ and $y' \in \mathcal{J}$, we have $\phi(a_1 y' + a' x) = (a_1, a')$ implying that ϕ is onto. Therefore, ϕ is an isomorphism i.e. $\frac{\mathcal{R}}{\mathcal{I}} \cong \frac{\mathcal{R}}{\mathcal{I}_1} \oplus \frac{\mathcal{R}}{\mathcal{J}}$. Inductively, we can show that $\frac{\mathcal{R}}{\mathcal{J}} \cong \frac{\mathcal{R}}{\mathcal{I}_2} \oplus \ldots \oplus \frac{\mathcal{R}}{\mathcal{I}_r}$ and hence, $\frac{\mathcal{R}}{\mathcal{I}} \cong \frac{\mathcal{R}}{\mathcal{I}_1} \oplus \ldots \oplus \frac{\mathcal{R}}{\mathcal{I}_r}$. ■

In $\mathbb{Z}$ (or $\mathbb{F}[x]$), two elements m_1 and m_2 are coprime integers (or polynomials) if and only if the ideals (m_1) and (m_2) are coprime. Applying the above theorem to the ring of integers (or polynomials) we get the following result.

Corollary 2 *Let $m \in \mathcal{R} = \mathbb{Z}$ (or $\mathbb{F}[x]$) be such that $m = \prod_{j=1}^{r} m_j$ where $m_1, \ldots, m_r$ are pairwise coprime integers (or polynomials). Then $\frac{\mathcal{R}}{(m)} \cong \frac{\mathcal{R}}{(m_1)} \oplus \ldots \oplus \frac{\mathcal{R}}{(m_r)}$.*

Thus every element of the ring $\frac{\mathcal{R}}{(m)}$ can be uniquely written as an r-tuple with the i^{th} component belonging to the ring $\frac{\mathcal{R}}{(m_i)}$. Addition and multiplication in $\frac{\mathcal{R}}{(m)}$ amounts to component-wise addition and multiplication in the rings $\frac{\mathcal{R}}{(m_i)}$. This suggests a strategy to speed up computation based on the fact that it is faster to compute modulo a small integer (or a small degree polynomial) than over integers (or polynomial ring).

- Given a bound, say A, on the output of a computation, choose small $m_1, \ldots, m_r$ such that $\prod_{i=1}^{r} m_i > A$ and do the computations modulo each m_i.
- At the end, combine the results of computations to get the desired result.

Let us see an application based on this idea next.

3 An application of CRT: Determinant computation

Suppose we want to compute the determinant of an $n \times n$ matrix M whose entries are integers. We can use Gaussian elimination, but it is not immediately clear whether the sizes of the numerators and denominators of the rational numbers appearing at intermediate stages of the elimination are polynomially bounded in the input size. (The input size is roughly between $n^2 - 1 + \log A$ and $n^2 \log A$ bits, where A is the maximum among the absolute values of the entries of M.) Is it possible that the intermediate expressions are very large? The answer to this is 'No'. But, the proof that the intermediate numbers do not swell up to a great extent during Gaussian elimination, is not quite easy. In today's class, we will see an alternative, *faster* approach to compute the determinant using Chinese remaindering. We say it is faster because CRT is inherently *parallelizable* and each computation happens over a small modulus.

Let A be the bound on the largest absolute value of the integer elements of M. Hadamard's inequality gives a bound on the absolute value of $\det(M)$ in terms of n and A.

Lemma 3 (Hadamard's Inequality) $|\det(M)| \leq n^{\frac{n}{2}} A^n$.

Proof Let $v_1, \ldots, v_n$ be the row vectors of M. Assuming that $v_1, \ldots, v_n$ are linearly independent (otherwise the determinant is zero), we can find an orthogonal basis $v_1^*, \ldots, v_n^*$, using Gram-Schimdt orthogonalization (see Appendix), such that $\|v_i^*\| \leq \|v_i\|$ for all i, $1 \leq i \leq n$. Here, $\|v\|$ denotes the 2-norm of vector v. It follows from the properties of this orthogonal basis (see Lemma 4 in Appendix) that,

$$\det(M)^2 = \det\left(M \cdot M^T\right) = \prod_{i=1}^{n} \|v_i^*\|^2 \leq \prod_{i=1}^{n} \|v_i\|^2 \leq n^n A^{2n}.$$

We use this bound in the following algorithm, which computes the determinant of M.

Algorithm 3 Computing determinant using Chinese remaindering

1. Let $B = n^{\frac{n}{2}} A^n$ and $r = \lceil \log(2B+1) \rceil$.
2. Let $m_1, \ldots, m_r$ be the first r primes and $m = \prod_{i=1}^{r} m_i$.
3. Compute $a_i = \det(M) \mod m_i$ for each i.
4. Compute α_i such that $\alpha_i \cdot \frac{m}{m_i} = 1 \mod m_i$ for each i.
5. Compute $d = \sum_{i=1}^{r} \alpha_i \cdot \frac{m}{m_i} \cdot a_i \mod m$.
6. If $d \leq B$ return d, else return $d - m$.

Correctness and time complexity - First note that $m > 2^r > 2B$. By the Prime Number Theorem, the r^{th} prime has value about $r \log r$, hence $m_i = O(r \log r)$, which means $m_1, \ldots, m_r$ can be found in $\tilde{O}(r^2)$

time (in Step 2). Computing a_i in Step 3 takes time that is polynomial in n and $\log A$, using Gaussian elimination over the field $\mathbb{F}_{m_i}$. Step 4 succeeds in finding an α_i as $\gcd(\frac{m}{m_i}, m_i) = 1$. Also, an α_i can be found in $\tilde{O}(r)$ time (why?). In step 5, d can also be computed in $\tilde{O}(r)$ time. From the choice of α_i it is clear that $d = a_i \bmod m_i$, for all i. By the Chinese remaindering theorem, $d = \det(M) \bmod m$. We know that $|\det(M)| \leq B < 2B < m$ (using Hadamard's inequality). Therefore, if $d \leq B$ then $\det(M) = d$, otherwise $\det(M) = d - m$. Therefore, the entire algorithm runs in time $\mathsf{poly}(n, \log A)$. We leave it as an exercise to find a precise expression for $\mathsf{poly}(n, \log A)$ in 'big-Oh' notation.

Exercises:

1. Show that in Step 2 Algorithm 1 there exist points $\mathbf{w}_1, \ldots, \mathbf{w}_k$ (independent of what the message is) such that the linear system in step 3 always has a solution.
2. Show that Reed-Muller codes are $(q-1, \delta, 2\delta/(1-\sigma))$-locally decodable, where $d \leq \sigma(q-1) - 1$, for all δ. (Infer that the decoder can tolerate nearly $1/4$ fraction of errors, while erring with probability less than $1/2$.).
3. In the time complexity analysis of Algorithm 3, find a precise expression for $\mathsf{poly}(n, \log A)$ using order notation.
4. Prove Lemma 4.

4 Appendix

4.1 Gram-Schmidt orthogonalization

Let $v_1, \ldots, v_n$ be linearly independent vectors in $\mathbb{R}^m$ and $\mathcal{V}$ be the space spanned by them. Gram-Schmidt orthogonalization is a technique to find orthogonal vectors $v_1^*, \ldots, v_n^*$ such that the space spanned by them is $\mathcal{V}$. We denote the dot product of two vectors u and w by $u \bullet w$ and $\|u\| = \sqrt{u \bullet u}$ is the 2-norm of u. The construction of the orthogonal vectors proceeds as follows,

$$\begin{aligned} v_1^* &= v_1 \quad \text{and} \\ v_i^* &= v_i - \sum_{j<i} \mu_{ij} v_j^* \quad \text{for } 2 \leq i \leq n \text{ where, } \mu_{ij} = \frac{v_i \bullet v_j^*}{v_j^* \bullet v_j^*} \text{ for } 1 \leq j < i. \end{aligned}$$

Define the projection matrix as $U = (\mu_{ij})_{1 \leq i,j \leq n}$ where $\mu_{ii} = 1$ for all i, $\mu_{ij} = 0$ for $j > i$ and $\mu_{ij} = \frac{v_i \bullet v_j^*}{v_j^* \bullet v_j^*}$ for $j < i$. Let V be the $n \times m$ matrix with $v_1, \ldots, v_n$ as the rows and V^* be the matrix with $v_1^*, \ldots, v_n^*$ as the rows. The following facts are easy to verify and are left as exercise.

Lemma 4 *1. The vectors $v_1^*, \ldots, v_n^*$ are mutually orthogonal (i.e. $v_i^* \bullet v_j^* = 0$ for $i \neq j$), and the space spanned by them is $\mathcal{V}$.*

2. The space spanned by $v_1, \ldots, v_i$ is the same as the space spanned by $v_1^*, \ldots, v_i^*$. Also, $\|v_i^*\| \leq \|v_i\|$ for all i.

3. $V = U \cdot V^*$.

4. $\det(U) = 1$ and so if $m = n$ then $\det(V) = \det(V^*)$.