Computational Number Theory and Algebra April 23, 2012

Lecture 2

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

In the last lecture, we discussed three applications, namely the RSA cryptosystem, Diffie-Hellman key exchange protocol and Reed-Solomon codes, where basic algebraic and number theoretic operations are used. In today's class, we will cover the following topics:

1. Decoding of Reed-Solomon codes,
2. Local decodability - e.g., Hadamard code

1 Decoding of Reed-Solomon codes

Decoding of Reed-Solomon codes comes in two flavors - unique decoding and list decoding. In unique decoding the decoder outputs only the message string, whereas in list decoding, the codeword is decoded to a small set of strings so that the original message is one among these strings. We will first discuss a version of the unique decoding algorithm due to Welch and Berlekamp [WB86].

1.1 Unique decoding of Reed-Solomon codes

Recall the Reed-Solomon encoding algorithm from the last lecture. We continue to use the notation introduced before: $P_m(x) \in \mathbb{F}_q$ is the message polynomial of degree $k-1$, $(c_0, \ldots, c_{n-1})$ is the codeword $(c_j = P_m(e_j)$ for $0 \leq j < n$ where $e_0, \ldots, e_{n-1}$ are distinct elements of $\mathbb{F}_q)$. Let $(d_0, \ldots, d_{n-1})$ be the received word which is possibly different from the codeword $(c_0, \ldots, c_{n-1})$. The following decoding algorithm finds the message polynomial $P_m(x)$ if the number of errors in $(d_0, \ldots, d_{n-1})$ is not too high.

Algorithm 1 Reed-Solomon codes: Unique decoding

1. Let $t = \lceil \frac{n-k+1}{2} - 1 \rceil$.
2. Find polynomials $M(x)$ and $F(x)$ such that,
   $\deg F(x) \leq t$, $F(x) \neq 0$;
   $\deg M(x) \leq k + t - 1$ and
   $M(e_i) = d_i \cdot F(e_i)$ for every $0 \leq i < n$.
3. Output $\frac{M(x)}{F(x)}$.

The correctness of the algorithm can be argued as follows. Let $\mathcal{E} \subset \{0, \ldots, n-1\}$ be the set of all positions in the received word $(d_0, \ldots, d_{n-1})$ which are in error i.e. $d_i \neq c_i$ for $i \in \mathcal{E}$. Assume that $|\mathcal{E}| \leq t$. Define the polynomials $E(x) \triangleq \prod_{i \in \mathcal{E}} (x - e_i)$ and $N(x) \triangleq P_m(x)E(x)$. It is easy to verify that $N(e_i) = d_i E(e_i)$ for all $0 \leq i < n$. Therefore, solutions for the polynomials $M(x)$ and $F(x)$ exist in step 2. Now notice that the polynomial $R(x) \triangleq N(x)F(x) - M(x)E(x)$ has degree at most $k + 2t - 1$ and $R(e_i) = 0$ for all $0 \leq i < n$. This implies that $R(x) = 0$ as $t < \frac{n-k+1}{2}$, and hence $\frac{M(x)}{F(x)} = \frac{N(x)}{E(x)} = P_m(x)$.

Step 2 can be implemented by solving a system of linear equations with the coefficients of $F(x)$ and $M(x)$ as unknowns. A preferable way of implementing this step is through rational function interpolation (for details see the online notes of Sudan [Sud] and Rosenblum [Ros]). In step 3 we can use polynomial division. Although very simple in nature, this algorithm does not achieve the best possible running time. The current fastest unique decoding algorithm is due to Justesen [Jus76].

The above decoding procedure can correct up to $\frac{n-k+1}{2}$ errors in the received word, and this error bound is the optimum as far as unique decoding is concerned (why?). However, it turns out that a lot more errors can be corrected if we allow the decoding procedure to output a small list of candidate message strings with the guarantee that the original message is one among them. This will be discussed next.

1.2 List decoding of Reed-Solomon codes

As before, $\mathcal{E} \subset \{0, \ldots, n-1\}$ is the set of all positions in the received word $(d_0, \ldots, d_{n-1})$ which are in error i.e. $d_i \neq c_i$ for $i \in \mathcal{E}$. The following is a simpler version of the list decoding algorithm given by Sudan [Sud97].

Algorithm 2 Reed-Solomon codes: List decoding

1. Let $u_0 = \lceil \sqrt{nk} \rceil$ and $u_1 = \lfloor \sqrt{\frac{n}{k}} \rfloor$.
2. Find a nonzero bivariate polynomial $Q(x, y)$ with x-degree u_0 and y-degree u_1 such that $Q(e_j, d_j) = 0$ for every $0 \leq j < n$.
3. Factor $Q(x, y)$.
4. Output all polynomials $P(x)$ such that $(y - P(x))$ is a factor of $Q(x, y)$.

We now show that the message polynomial $P_m(x)$ is in the output list of polynomials if $|\mathcal{E}| < n - 2\lceil \sqrt{nk} \rceil$ ($\mathcal{E}$ is the set of 'error' indices). In step 2 we can find $Q(x, y)$ by solving a system of linear equations with the coefficients of $Q(x, y)$ as variables. Since there are $(1 + u_0)(1 + u_1) > n$ unknowns with n equations to satisfy, there always exists a solution for $Q(x, y)$. Consider the polynomial $S(x) \triangleq Q(x, P_m(x))$. Degree of $S(x)$ is strictly less than $u_0 + u_1 k \leq 2\lceil \sqrt{nk} \rceil$. If $|\mathcal{E}| < n - 2\lceil \sqrt{nk} \rceil$ then from step 2 it follows that $S(e_j) = 0$ for at least $2\lceil \sqrt{nk} \rceil$ many distinct e_j's. This implies that $S(x) = 0$ and hence $(y - P_m(x))$ is a factor of $Q(x, y)$.

Another fundamental algebraic operation, namely polynomial factoring, is used in step 3 of the list decoding algorithm. In this case, it is bivariate polynomial factoring.

In practice, the value of k is much smaller than n and hence $n - 2\lceil \sqrt{nk} \rceil$ is much larger than $\frac{n-k+1}{2}$ (say, when $k \leq n/16$). The current best parameter is due to Guruswami and Sudan [GS99] that can correct up to $n - \sqrt{nk}$ errors, which is perhaps the best possible (see [BSKR06]). Although, Reed-Solomon codes tolerate an optimum number of errors, one drawback of this family of codes is that the underlying alphabet set $\mathbb{F}_q$ needs to be as large as the codeword length n, i.e. $q > n$. There are known code constructions that avoid this problem - for example, BCH codes. For an excellent exposition to the theory of error-correcting codes the reader may refer to the PhD thesis of Guruswami [Gur04] or the book by MacWilliams and Sloane [MS81].

2 Local decodability - e.g., Hadamard code

Motivation - Imagine a situation where the data one needs to encode is too long, for instance it could be information about several hundreds of clients. One may choose to split the data into shorter messages, say one message per client, and then encode these messages individually. However, that way if one codeword gets entirely corrupted then information about a client is completely lost. To accommodate a larger fraction of errors, it seems better to encode the whole data (i.e information of all the clients) as one single message into a single 'large' codeword. This way, the codeword tolerates a much larger fraction of errors without losing information. But, there is an issue here: Whenever a client wants to access his/her information, he/she needs to decode the entire 'large' codeword just to obtain his/her 'tiny' fraction of information. The class of codes which addresses this issue of retrieving part of the message, *without* looking at the whole codeword is known as *locally decodable codes*. Reed-Muller codes belong to this class of codes - we will discuss this in the next class. In today's class, we will see the example of Hadamard codes. Locally decodable codes have many

important applications in private information retrieval, secure multiparty communication and average-case complexity. Refer to the survey [Yek12] to know much more about locally decodable codes.

2.1 Hadamard codes

Suppose that we have a binary message string $m = (m_1, \ldots, m_k)$, where $m_i \in \{0, 1\}$. The codeword corresponding to m is a 2^k length binary string $\mathcal{C}$ whose bit positions are indexed by all subsets of $\{1, \ldots, k\}$. Denote the bit of $\mathcal{C}$ which is indexed by $S \subseteq \{1, \ldots, k\}$, as $\mathcal{C}_S$. The encoding scheme is as follows: $\mathcal{C}_S = \oplus_{j \in S} m_j$, where $\oplus$ denote the *xor* of bits. Suppose that at most δ fraction of the bits in $\mathcal{C}$ are corrupted. $\mathcal{D}$ is the (possibly corrupted) received word. Let's say, we want to decode the i^{th} bit of the message, i.e., find m_i. The decoding process is as follows: Pick a subset $S \subseteq \{1, \ldots, k\}$ uniformly at random, and output $\mathcal{D}_S \oplus \mathcal{D}_{S\Delta\{i\}}$, where Δ denotes the *symmetric difference* of sets. The following claims show that the output is m_i with high probability. The proof of the claims are left as exercise.

Claim 1 *If $S \subseteq \{1, \ldots, k\}$ is chosen uniformly at random then $S\Delta\{i\}$ is also a random subset of $\{1, \ldots, k\}$ for any fixed i. Meaning, for any fixed $T \subseteq \{1, \ldots, k\}$, $\Pr_S\{S\Delta\{i\} = T\} = 1/2^k$.*

Claim 2 *$\mathcal{D}_S = \mathcal{C}_S$ with probability at least $1 - \delta$. Similarly, $\mathcal{D}_{S\Delta\{i\}} = \mathcal{C}_{S\Delta\{i\}}$ with probability at least $1 - \delta$. Therefore, $\mathcal{D}_S \oplus \mathcal{D}_{S\Delta\{i\}} = \mathcal{C}_S \oplus \mathcal{C}_{S\Delta\{i\}} = m_i$ with probability at least $1 - 2\delta$.*

Thus, by reading only *two* bit positions of the 2^k-long received word, we can infer the correct value of any fixed bit of the message with high probability (provided the fraction of errors δ is *small*, say 1/5). However, the problem with Hadamard codes is that the codeword length is exponential in the message length. We will see a better construction in the form of Reed-Muller codes in the next class.

A list of basic operations we have come across so far

We have seen quite a few basic operations being used in the applications discussed so far. These are: primality testing, integer multiplication, gcd computation, modular inverse, modular exponentiation, integer factoring, discrete logarithm, multipoint evaluation, solving a system of linear equation, polynomial factoring (over finite fields) and polynomial interpolation. Apart from these we have also noticed two very useful objects in algebraic computation - the use of multivariate polynomials, and the use of randomness.

The design and analysis of algorithms for the basic operations involve many other fundamental mathematical results or *tools*. We will discuss about several such tools during this course. The tool we discuss in the next lecture is Chinese remaindering theorem.

Exercises:

1. Prove Claims 1 and 2.

References

[BSKR06] Eli Ben-Sasson, Swastik Kopparty, and Jaikumar Radhakrishnan. Subspace Polynomials and List Decoding of Reed-Solomon Codes. In *FOCS*, pages 207–216, 2006.

[GS99] Venkatesan Guruswami and Madhu Sudan. Improved decoding of Reed-Solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, 1999.

[Gur04] Venkatesan Guruswami. *List Decoding of Error-Correcting Codes (Winning Thesis of the 2002 ACM Doctoral Dissertation Competition)*, volume 3282 of *Lecture Notes in Computer Science*. Springer, 2004.

[Jus76] Jørn Justesen. On the complexity of decoding Reed-Solomon codes. *IEEE Transactions on Information Theory*, 22(2):237–238, 1976.

[MS81] Florence Jessie MacWilliams and Neil J. A. Sloane. *The Theory of Error-Correcting Codes*. North Holland, Amsterdam, 1981.

[Ros] Michael Rosenblum. A fast algorithm for rational function approximations. Available from `http://people.csail.mit.edu/madhu/FT01/notes/rosenblum.ps`.

[Sud] Madhu Sudan. Notes on an efficient solution to the rational function interpolation problem. Available from `http://people.csail.mit.edu/madhu/FT01/notes/rational.ps`.

[Sud97] Madhu Sudan. Decoding of Reed Solomon Codes beyond the Error-Correction Bound. *J. Complexity*, 13(1):180–193, 1997.

[WB86] Lloyd R. Welch and Elwyn R. Berlekamp. Error correction for algebraic block codes, December 1986. U.S. Patent Number 4,633,470.

[Yek12] Sergey Yekhanin. Locally Decodable Codes. *Foundations and Trends in Theoretical Computer Science (to appear)*, 2012. Available from `http://research.microsoft.com/en-us/um/people/yekhanin/Papers/LDC_now.pdf`.