MTHSC 985 Fall 2001 Clemson University

Fast Fourier transforms and
sparse linear systems over finite fields Instructor: Shuhong Gao

Lecture 1, August 23 & 28 Scribe: Virgínia Rodrigues

1. Sparse linear systems in practice

In this course we shall focus on linear systems over finite fields. These systems arise in many applications including coding theory, cryptography, and symbolic computation where exact arithmetic is desirable.

Some of the methods that we shall discuss are adapted from numerical methods, but when computing over finite fields we have different issues to deal with. In numerical computation (over real numbers), numerical stability due to round-off errors is a major concern. In computation over finite fields there are no round-off errors, but instead there exist self-orthogonal vectors (called *isotropic vectors*), which cause a different set of issues in algorithm design. Our goal in this course is to present the basic theory and the best algorithms known for solving sparse linear systems over finite fields. Analysis of these algorithms will be a major concern and we hope to solve some of the problems that are still open at the moment.

In this lecture, we shall delineate various kinds of linear systems that arise in practice and mention briefly the major issues on which we will focus in this course.

1. Coding Theory. In decoding (n, k, d) Reed-Solomon codes and BCH codes (and many others indeed), one needs to solve a key equation over finite fields; namely, given a polynomial $s(x) \in \mathbb{F}_q[x]$ of degree $2t = d - 1$, find polynomials $\omega(x), \sigma(x) \in \mathbb{F}_q[x]$ such that

$$\omega(x) \equiv s(x)\sigma(x) \pmod{x^d}, \tag{1}$$

where $\deg \omega(x) \leq t - 1$, $\deg \sigma(x) \leq t$. Here t bounds the number of errors in a received vector, $s(x)$ is the syndrome polynomial, $\sigma(x)$ is the error locator polynomial, and $\omega(x)$ is the error evaluator polynomial.

Certainly (1) is a linear system for the coefficients of $\omega(x)$ and $\sigma(x)$. We know from last semester that it can be solved by applying the extended gcd algorithm to $s(x)$ and x^d. It can also be solved by Berlekamp's algorithm. These algorithms are equivalent and require $\mathcal{O}(d^2)$ operations in $\mathbb{F}_q$. It is natural to ask if it is possible to solve (1) faster. We shall show in this course that it can be solved in time $\mathcal{O}(d \log d)$ for special fields $\mathbb{F}_q$ and $\mathcal{O}(d \log d \log\log d)$ in general.

We can write down the equation for the coefficients of $\sigma(x)$ explicitly. Suppose $s(x) = s_0 + s_1 x + s_2 x^2 + \cdots + s_{d-1} x^{d-1}$ is given. Let $\sigma(x) = a_0 + a_1 x + \cdots + a_t x^t$. Then

$$s(x)\sigma(x) = \cdots + \sum_{k=t}^{d-1} \left(\sum_{i=0}^{t} s_{k-i} a_i \right) x^k + x^d(\cdots).$$

Since $\deg w(x) \leq t-1$, equation (1) implies that

$$\sum_{i=0}^{t} s_{k-i} a_i = 0, \quad t \leq k \leq 2t = d-1,$$

that is

$$\begin{pmatrix} s_0 & s_1 & \dots & s_t \\ s_1 & s_2 & \dots & s_{t+1} \\ \vdots & \vdots & \ddots & \vdots \\ s_t & s_{t+1} & \dots & s_{2t} \end{pmatrix} \begin{pmatrix} a_t \\ a_{t-1} \\ \vdots \\ a_0 \end{pmatrix} = 0. \tag{2}$$

The matrix in (2) is called a *Hankel matrix*. Solving (2) and (1) is essentially equivalent.

Computing the syndrome polynomial $s(x)$ is also equivalent to solving a linear system. In decoding cyclic codes, $s(x)$ is computed as the remainder of a polynomial $r(x)$ (corresponds to a received vector) by a fixed polynomial $h(x)$ (which defines the code), i.e.

$$r(x) = q(x) \cdot h(x) + s(x), \tag{3}$$

where $\deg r(x) = n-1, \deg q(x) = k-1, \deg h(x) = n-k$ and $\deg s(x) \leq n-k-1$.

The long division algorithm finds $s(x)$ in time $\mathcal{O}(k(n-k))$. We ask for more: *is it possible to compute the remainder $s(x)$ in time $\mathcal{O}(n \log n)$?*

Gao recently discovered a new algorithm for decoding RS codes. This algorithm has three steps. In one step it is desirable to find $q(x)$ in (3), where the polynomials involved have different degrees and $h(x)$ typically has a small degree (bounded by the number of errors). Another step is similar to (1):

$$\omega(x) \equiv s(x)\sigma(x) \pmod{f(x)}, \tag{4}$$

where $f(x)$ is fixed with degree n (e.g. $f(x) = x^q - x$ if $n = q$), $s(x)$ is given with $\deg s(x) \leq n-1$, and $\omega(x)$ and $\sigma(x)$ are unknown with $\deg \omega(x) \leq k+t-1$, and $\deg \sigma(x) \leq t$. *How to solve (4) in time $\mathcal{O}(n \log n)$?*

The remaining step (which is the first step of the algorithm) is to compute $s(x)$. Suppose that $a_1, \dots, a_n \in \mathbb{F}_q$ are fixed and let $(b_1, \dots, b_n) \in \mathbb{F}_{q^n}$ be any received vector. Then $s(x)$ is the unique polynomial in $\mathbb{F}_q[x]$ such that

$$s(a_i) = b_i, \ 1 \leq i \leq n. \tag{5}$$

The problem of finding $s(x)$ is called *interpolation* or *inverse Fourier transform*. *How fast can one interpolate?* We shall show how to solve this problem in time $\mathcal{O}(n \log n)$ for some special cases.

The problems (1)-(5) are not just important in coding theory, but fundamental by themselves, since fast algorithms for division, gcd and interpolation are basic procedures in many applications.

2. Cryptography. Many public-key cryptosystems are based on two hard problems in number theory, namely, factoring integers and computing discrete logs over finite fields. The fastest algorithms known for solving these problems are based in index calculus methods using number field sieve or function field sieve. The basic idea is to first find a large collection of relations (on elements of a factor base) and then combine them to obtain factors or discrete logs. The last step amounts to solving a large linear system over finite fields.

For factoring integers, the linear system is of the form

$$Ax \equiv 0 \pmod 2, \tag{6}$$

where A is an $n \times m$ matrix with entries 0 or 1, and $m > n$. Each column of A corresponds to a relation (of elements in a factor base) and it is desirable to find one or several nonzero solutions x.

The matrix A in (6) is extremely sparse. For example, in August 1999, RSA-155 (an integer with 155 digits) was successfully factored by an international team of researchers. For the matix A involved,

$$\begin{aligned} n &= 6,699,191, \\ m &= 6,711,336 \end{aligned}$$

and on average each row of A has only 63 nonzero entries. Solving (6) via Gauss elimination requires $\mathcal{O}(n^3)$ bit operations and storage of $\mathcal{O}(n^2)$ bits. For $n = 6,000,000$, storing n^2 bits requires more than 4000 GB of memory, which is out of the capacity of most, if not all, existing computers. *How did they solve the system?*

In computing discrete logarithm, the linear system to be solved is of the form

$$Ax \equiv b \pmod r, \tag{7}$$

where r is a large prime, A is an $n \times m$ matrix with integral entries and b is a nonzero vector. Here A is typically sparse, $n > m$ and the solution is likely to be unique.

In the US Digital Signature Standard (DSS) r is about 2^{160}, and in other public-key cryptosystems r is typically about $2^{1,000}$. The matrix A will also have millions of columns and rows. *How to solve such systems fast?*

3. Factoring polynomials. Factoring polynomials is a fundamental routine in any computer algebra system (e.g. Maple, Mathematica, Magma, etc). Most efficient algorithms for factoring polynomials need first to solve a linear system and then use the solutions to get factors of the polynomial to be factored. Let us mention two methods: Berlekamp's for univariate polynomials and Gao's for bivariate polynomials.

Suppose $f \in \mathbb{F}_q[x]$, squarefree, of degree n, is given to be factored. Berlekamp's method needs to find $g(x) \in \mathbb{F}_q[x]$ of degree smaller than n such that

$$g(x)^q \equiv g(x) \pmod{f(x)}. \tag{8}$$

Since q-th power is a linear operation on the coefficients of $g(x)$, (8) is indeed a linear system and it is desirable to find a basis for all the solutions (or several random solutions). When $f(x)$ has high degree, (8) is a large linear system to solve over $\mathbb{F}_q$.

To see the equation from Gao's PDE method, let $f \in \mathbb{F}[x, y]$, squarefree, with bidegree (m, n) (i.e. degree m and n in x and y, respectively). One needs to find polynomials $g, h \in \mathbb{F}[x, y]$ such that

$$\frac{\partial}{\partial y}\left(\frac{g}{f}\right) = \frac{\partial}{\partial x}\left(\frac{h}{f}\right), \tag{9}$$

where $\deg g \leq (m-1, n)$ and $\deg h \leq (m, n-1)$. The equation (9) can be rewritten as

$$f \cdot \left(\frac{\partial g}{\partial y} - \frac{\partial h}{\partial x}\right) - g \cdot \frac{\partial f}{\partial y} + h \cdot \frac{\partial f}{\partial x} = 0. \tag{10}$$

It is desirable to find a basis for all the solutions g (or several random solutions g). Again, *how to solve (10) fast?*

4. Solving nonlinear systems. Finding solutions of a system of nonlinear polynomial equations over a given field is fundamental in symbolic and scientific computations. Applications include computer algebra, computer graphics, computer vision, geometric and solid modeling, economics and optimization, molecular biology, robotics, etc. This is an extremely active area of current research.

A system of nonlinear polynomial equations can also be turned into a linear system, usually sparse. We illustrate by a few examples.

Let $\mathbb{F}$ be any field and $f = x^n + \sum_{i=0}^{n-1} a_i x^i \in \mathbb{F}[x]$. We want to find the zeros of $f(x)$ in $\mathbb{F}$. Let

$$A = \begin{pmatrix} 0 & 1 & & & \\ & 0 & 1 & & \\ & & \ddots & \ddots & \\ & & & 0 & 1 \\ -a_0 & -a_1 & \dots & -a_{n-2} & -a_{n-1} \end{pmatrix},$$

the companion matrix of f. Then $\lambda \in \mathbb{F}$ is a zero of f iff

$$A \cdot v = \lambda \cdot v \tag{11}$$

for some nonzero vector $v \in \mathbb{F}^n$, that is, λ is an eigenvalue of A (that lies in $\mathbb{F}$).

More generally, let $f_1, \dots, f_m \in \mathbb{F}[x_1, \dots, x_n]$ form a Gröbner basis for the ideal $\mathbf{I} = \langle f_1, \dots, f_m \rangle$ with a monomial basis B. Suppose the ideal $\mathbf{I}$ is radical and has only finitely many common zeros, say $\{P_1, \dots, P_t\}$ in $\overline{\mathbb{F}}$. So B is finite and form a basis for the quotient ring

$$\mathbf{R} = \mathbb{F}[x_1, \dots, x_n]/\mathbf{I}.$$

Let $g \in \mathbf{R}$ be any element. Then multiplication by g in $\mathbf{R}$ defines a linear map:

$$\begin{array}{rccl} g: & \mathbf{R} & \longrightarrow & \mathbf{R} \\ & u & \longmapsto & g \cdot u \pmod{\mathbf{I}} \end{array}$$

This map can be computed via the monomial basis B. The eigenvalues of the linear map g are exactly the values of g at the points $P_1, \ldots, P_t$. Finding the common zeros of f is equivalent to finding the eigenvalues (or eigenvectors) of the linear map g, for various choices of g.

In practice, the given polynomials $f_1, \ldots, f_m$ may not be so nice (in that they may not form a Gröbner basis). One may first try to construct a Gröbner basis, but this process may be prohibitively time consuming. In this case, one can use resultant methods to convert a nonlinear system into a linear system. (Include a nice example here ...)

5. General Model: black box approach. Let V be a vector space of dimension n over a field $\mathbb{F}$. Let T be a linear operator on V and $b \in V$. The problem is to find $x \in V$ such that

$$Tx = b. \tag{12}$$

When $b = 0$ we require $x \neq 0$.

Typically, we shall think of T as a matrix. In the above applications, T could be given explicitly as a sparse matrix (as in (6) and (7)) or given implicitly. In the first case, the matrix-vector product $T \cdot v$ can be computed fast in a straightforward way. In other applications, T may not look sparse (if written out explicitly), but the matrix-vector product $T \cdot v$ can also be computed quickly via fast multiplication of polynomials, in this case we also say that T is *sparse*. In general, we take the point of view of Kaltofen and Trager (1990), that is, we simply view T as a *black box*, or an *oracle*, that returns the product $T \cdot v$ for any given vector v. Pictorially, we have

FIGURE 1. Black box for T

In a specific application, one has to design this black box, that is, find a fast algorithm for computing the matrix-vector product $T \cdot v$. When solving (12), one is given the black box but not allowed to "peek inside" the box. The main goal in designing efficient algorithms for solving (12) is to call the black box as few times as possible and use extra storage (for the intermediate vectors) as less as possible.