| | |
|---|---|
| **6.966 Algebra and Computation** | October 7, 1998 |
| Lecture 8 | |
| *Lecturer: Madhu Sudan* | *Scribe: Venkatesan Guruswami* |

In this lecture we will wrap up the algorithm presented in the previous lecture for bivariate polynomial factoring. We will then present some thoughts on multivariate factoring which we will see in detail in the next lecture. Finally, we will digress a little bit and present an application to decoding Reed-Solomon codes.

# 1 Bivariate Polynomial Factoring

We first recall the skeleton of the algorithm that we discussed in the last lecture. The idea is to use univariate polynomial factoring over any U.F.D for bivariate factoring. Recall that if $R$ is a U.F.D then so is $R[x_1, x_2, \ldots, x_n]$ for any $n \geq 1$, so the factorization will be unique up to associates. Also we only seek to find some non-trivial factor, since then we can recursively compute the complete factorization.

Algorithm Bivariate-Factor:
Input: A polynomial $f \in R[x, y]$, $R$ a U.F.D, with $\deg_x(f) \leq d$, $\deg_y(f) \leq d$.
Output: Some non-trivial monic factor $g \in R[x, y]$ of $f$.

1. Preprocess $f$ appropriately – this step will be expanded soon.
2. Factor $f_0(x) = f(x, 0)$ as $f_0(x) = g_0(x)h_0(x)$ where $g_0, h_0$ are relatively prime and $g_0$ is irreducible (This is possible assuming $f_0$ has no repeated factors, and we will see how to remove this assumption soon). Equivalently, we can view this factorization as $f(x, y) \equiv g_0(x, y)h_0(x, y)(\mod y)$. Repeatedly apply Hensel lifting $K$ times, for $K = 2\lceil \log d \rceil + 2$, to find $g_K, h_K$ such that
$$f(x, y) \equiv g_K(x, y)h_K(x, y) \quad (\mod y^{2^K}).$$
3. Solve the *linear system* (with unknowns being the coefficients of $G$, $L$) to find $G$ and $L$ with $\deg_x(G), \deg_x(L) < \deg_x(f)$, and $\deg_y(G) \leq \deg_y(f)$ $\deg_y(l) \leq 2^K$; that satisfy
$$G(x, y) \equiv g_K(x, y)L(x, y) \quad (\mod y^{2^K}).$$
4. Find $\gcd(G, f)$, and if non-trivial output $G$ as it as a factor of $f$. If it is trivial, output that $f$ is irreducible.

We saw in the previous lecture that the Steps 2,3,4 all work if we assume that $f_0$ has no repeated factors. We can as usual, by a preliminary gcd computation like $\gcd(f, \frac{\partial f(x,y)}{\partial x})$, assume that $f$ has no repeated factors, but that will not suffice – for instance when $f(x, y) = x^3 + y(y + 1)$ (which is irreducible and hence definitely square-free), $f_0(x) = x^3$, which has repeated factors. The idea to fix this is to consider "shift" $y$ by a value $\beta \in R$ – specifically, consider $f_\beta(x, y) \stackrel{\Delta}{=} f(x, y + \beta)$ for $\beta \in R$. Clearly, if we factor $f_\beta(x, y)$ then we can recover the factors of $f(x, y)$ as well. We hope to find (and work with) a $\beta$ such that $f_\beta(x, 0)$ $(= f(x, \beta))$ has no repeated factors as a polynomial in $R[x]$.

**Lemma 1** *If $|R| > 4d^2$, then for any polynomial $f(x, y) \in R[x, y]$ with $\deg_x(f) \leq d$ and $\deg_y(f) \leq d$, there exists a $\beta \in R$ such that $f_\beta(x, 0) \in R[x]$ has no repeated factors.*

**Proof:** Consider the discriminant $\mathrm{disc}_x(f)$ treating $f$ as belonging to $(R[y])[x]$. It is clearly a polynomial $D(y)$ in $y$ of degree at most $(2d)^2 = 4d^2$. Hence we can find $\beta \in R$ that is not a root of $D$, and for such a $\beta$, we have $\mathrm{Disc}(f_\beta(x, 0)) \neq 0$, implying $f_\beta(x, 0)$ has no repeated factors. □

Thus the preprocessing step 1 in algorithm Bivariate-Factor expands as follows:

**1a.** Obtain square-free factorization of $f$ and work with each square-free factor separately. +

**1b.** Find $\beta \in R$ such that $f(x, \beta)$ is a square-free polynomial in $R[x]$.

**1c.** Let $f_\beta(x, y) \triangleq f(x, y+\beta)$, and work with $f_\beta$ (and at the end of the algorithm recover factors of $f$ from those of $f_\beta$).

**Corollary 2** *For any U.F.D $R$ with $|R| > 4d^2$, then factorization of degree $d$ polynomials in $R[x, y]$ can be reduced to univariate factorization in $R[x]$.*

In particular the above implies we can always reduce factorization in $R[x_1, x_2, \ldots, x_n]$ to factorization in $R[x_1, x_2, \ldots, x_{n-1}$ and inductively all the way down to factorization in $R[x_1, x_2]$.

We still have to deal with the issue of small values of $|R|$. But note that if $|R|$ is small, it is in particular finite, and hence $R$ is in fact a field. Now given $f \in R[x, y]$, we work with a finite extension $R'$ of $R$ such that $|R'| > 4d^2$ and obtain a non-trivial irreducible factor (if any) $g' \in R'[x, y]$ of $f \in R'[x, y]$ using the above corollary. We now need to recover a factor $g \in R[x, y]$ using $g' \in R'[x, y]$.

**Lemma 3** *Let $g', f$ be as above. Let $g \in R[x, y]$ be the minimal polynomial such that $g'(x, y)|g(x, y)$ (in $R'[x, y]$). Then $g(x, y)|f(x, y)$.*

**Proof:** Since $g'$ is irreducible, so is the minimal polynomial $g \in R[x, y]$ such that $g'|g$. Now if $g$ does not divide $f$, then $\exists\ u_x, v_x \in R[x, y]$ and $p \in R[x]$ (resp. $u_y, v_y \in R[x, y]$ and $q \in R[y]$) such that $u_x(x, y)f(x, y) + v_x(x, y)g(x, y) = p(x)$ (resp. $u_y(x, y)f(x, y) + v_y(x, y)g(x, y) = q(y)$). But then $g'(x, y)$ divides both $p(x)$ and $q(y)$ which is impossible (since $g'$ is not a constant polynomial). $\square$

We can thus recover a factor $g$ of $f$ in $R[x, y]$ using $g'$ as follows (we give only an informal description to illustrate the idea – details can be easily filled in). For every possible values $d_x, d_y$ (try them in increasing order), try to find a polynomial $h \in R'[x, y]$ with $\deg_x(h) = d_x$ and $\deg_y(h) = d_y$ such that $g \cdot h' = g \in R[x, y]$. This can be clearly viewed as a linear system over $R'$ with the added condition that the coefficients of $g$ come from $R$. But since $R'$ is a finite field extension of $R$, we can treat it as $R^l$ for some $l$, and then it is easy to transform the above requirements into a linear system over $R$. Indeed, assume the original system is $Ax = y$ with $A$ being a $r \times n$ matrix over $R'$, and we seek solutions $x \in {R'}^n$ and $y \in R^r$. We can rewrite this system as the homogeneous system $Bz = \bar{0}$ over $R'$ where $B = [A| - I_r]$ is a $r \times n + r$ matrix and we seek solutions $z_1, z_2, \ldots, z_{n+r} \in {R'}^{n+r}$ with $z_{n+i} \in R$ for $1 \leq i \leq r$. Since $R'$ is a finite field extension of $R$, it can be viewed as the vector space $R^l$, with a basis $\{\alpha_1, \alpha_2, \ldots, \alpha_l\}$ where $\alpha_i \in R'$ for $1 \leq i \leq l$ and $\alpha_1 = 1 (\in R)$. Hence we can replace each $z_i$, $1 \leq i \leq n$, by $l$ variables $z_{i1}, z_{i2}, \ldots, z_{il}$ that take values in $R$, and form a $rl \times (nl + r)$ matrix $C$ over $R$ by replacing each entry $b_{ij} \in R'$ of the matrix, for $1 \leq j \leq n$, by the $l \times l$ matrix over $R$ to which $b_{ij}$ corresponds when viewed as a linear transformation of $R'$ (over the basis $\{\alpha_1, \ldots, \alpha_l\}$ for $R'$ over $R$), and replacing column $n + i$ of $B$, for $1 \leq i \leq r$, by the vector which is zero everywhere except for a $(-1)$ in row $1 + (i - 1)l$. This gives a homogeneous linear system $Cz' = \bar{0}$ over $R$ solving which will give us the original polynomials we sought (after tracing back through the above conversions).

# 2 Multivariate Factoring

We now present some thoughts on extending this to multivariate polynomials. The first natural method for this is to use the reduction of Corollary 2 repeatedly starting from $n$ variables down to $n - 1$ and so on till we reach two variables when we can use the previous algorithm. Note that in all these steps the underlying U.F.D is a polynomial ring and hence is infinite and in particular the condition $|R| > 4d^2$ will be satisfied. This procedure, however, is clearly not very efficient. An alternative procedure is sketched below.

Let $f \in R[x, y_1, y_2, \ldots, y_n]$ be a polynomial that has to be factored. As in the bivariate case, we first factor $f$ (using univariate factorization) as

$$f(x, y_1, y_2, \ldots, y_n) \equiv g_0(x, y_1, y_2, \ldots, y_n) \cdot h_0(x, y_1, y_2, \ldots, y_n) \pmod{(y_1, y_2, \ldots, y_n)}.$$

Define $p \triangleq f - g_0 h_0$; then $p \in (y_1, y_2, \ldots, y_n)$. As we did for the bivariate case, we once again perform repeated Hensel-lifting up to a certain number $k$ of times, working with the ideal generated by $p$ (i.e in the $j$th step we find $g_j, h_j$ such that $f \equiv g_j \cdot h_j (\mod p^{2^k})$. The rest of the argument will also be analogous to the bivariate case, for instance instead of computing resultants over $R(y)$ (as was done in the case of $R[x, y]$) we will now compute resultants over $R(y_1, y_2, \ldots, y_n)$, etc. We omit the details which can be easily filled in.

The second approach is more efficient than the first one we suggested, but still both the methods perform poorly for one non-circumventable (at least apparently) reason. Even though the original multivariate polynomial may be sparse and hence have small description, it may very well be the case that one of its factors in not sparse. And since a degree $d$ polynomial on $n$ variables can have up to $\binom{n+d}{d}$ non-zero coefficients, the run-time of these methods in general grows as $O(n^d)$ even if $f$ itself has a small description. This is unavoidable for any algorithm that hopes to present an explicit representation of the factors in terms of their coefficients. The way around this is to ask for an algorithm that when given a degree $d$ polynomial on $n$ variables $f = \prod_{i=1}^{k} f_i$ (the $f_i$'s not necessarily distinct) as a black-box (i.e the algorithm has *oracle access* to the function $f$), must produce $k$ black-boxes $O_1, O_2, \ldots, O_k$ where each $O_i$ computes $f_i$ by (possibly) making queries to the oracle for $f$, and such that on any $\bar{a}$, $O_i$ computes $f_i(\bar{a})$ efficiently by making poly$(n, d)$ queries to the oracle for $f$. We will describe an algorithm by Kaltofen that achieves this in the next lecture. We now take a small digression to discuss a decoding algorithm for Reed-Solomon codes.

# 3 Application: Decoding Reed-Solomon Codes

Recall that in a $[n, k+1, n-k]_q$ Reed-Solomon code the message is given by degree $k$ polynomials in $F_q[X]$ and the codewords are obtained by evaluating the message polynomials at $n$ *distinct* points $x_1, x_2, \ldots, x_n \in F_q$ (so we need $q \geq n$). The distance of the code id $(n-k)$ since no two distinct degree $k$ polynomials over a field can agree at more than $k$ points.

Suppose a message $p \in F_q[X]_{\leq k}$ was thus encoded and transmitted, and we receive $\vec{y} = y_1, y_2, \ldots, y_n \in F_q^n$, with, say at most $e$, errors in transmission. We now wish to decode $\vec{y}$ and "reconstruct" $p$. If $e < \frac{n-k}{2}$, then we know that at most one such polynomial $p$ exists. We now describe an algorithm to find the polynomial $p$ in such a case.

Let $\mathcal{E} = \{i : p(x_i) \neq y_i\}$ so that $\mathcal{E}$ is the set of positions that are in error. We define the *error-locator polynomial* $E(x) = \prod_{i \in \mathcal{E}} (x - x_i)$ – we know that $\deg(E) \leq e$. Now for every $i$, $1 \leq i \leq n$, one of $y_i = p(x_i)$ or $E(x_i) = 0$ holds. Hence if we set $M(x) = p(x)E(x)$, then $\deg(M) \leq e + k$ and $E(x_i)y_i = M(x_i)$ for $1 \leq i \leq n$.

This motivates the following algorithm:

Algorithm Unique-Decode-RS-codes:
Input: A set of $n$ pairs $\{(x_i, y_i) \in F_q^2 : 1 \leq i \leq n\}$ such that $\exists$ a polynomial $p \in F_q[X]$ of degree at most $k$ such that $p(x_i) \neq y_i$ for at most $e$ values of $i$. (with $e < (n-k)/2$)
Output: The polynomial $p$.

1. Find polynomials $F, N \in F_q[X]$, $\deg(F) \leq e$, $\deg(N) \leq e + k$, $F \neq 0$, such that $F(x_i)y_i = N(x_i)$ for $1 \leq i \leq n$.
2. Output $p(x) = \frac{N(x)}{F(x)}$. /* See the remark below for an alternative, and in fact more preferred, implementation of this step. */

To prove the algorithm works, we first note the linear system in Step 1 above has a solution – indeed the polynomials $E, M$ from our discussion preceding the algorithm form a solution. A solution $(F, N)$ can thus be found by solving the linear system.

We now argue that for any $F, N$ found in Step 1, we will have $N(x) = F(x)p(x)$. To prove this, set $S = \{i : p(x_i) = y_i\}$, by hypothesis $|S| \geq n - e$. Also for every $i \in S$, $N(x_i) = F(x_i)p(x_i) = 0$. Thus the polynomial $N(x) - F(x)p(x)$ has at least $n - e$ roots, while its degree is at most $e + k$. Hence if $e + k < n - e$, the polynomial will be identically zero implying $p(x) = \frac{N(x)}{F(x)}$. But this condition is precisely $e < \frac{n-k}{2}$ which is given to us. The algorithm is thus guaranteed to work.

**Remarks on Run-time of the Algorithm**

- Solving the linear system $F(x_i)y_i = N(x_i)$ can be done in $O(n\text{poly}(\log n))$ time using the special form of the system.
- Often a much faster way to obtain the message polynomial $p(x)$ is to compute the (at most $e$) roots of $F(x)$ using a root-finding algorithm, and to then obtain $p(x)$ by fast interpolation at (any of) the points $x_i$ which are *not* roots of $F(x)$. Since the degree of $F$ is at most $e$, and $e$ is often very small, this method is in practice much faster. In fact, this was one of the primary motivations to develop fast algorithms for root-finding of univariate polynomials.