Error Correcting Codes: Combinatorics, Algorithms and Applications **(Fall 2007)**

Lecture 38: Guruswami-Sudan List Decoder

November 11, 2007

Lecturer: Atri Rudra *Scribe: Sandipan Kundu & Atri Rudra*

1 General Structure of RS list decoder

We recall the structure of a "generic" list decoder for RS codes. Given the received word $(\alpha_i, y_i) \in (\mathbb{F}_q^2)^n$,

- **Step1**: Find a non-zero $Q(X, Y)$ with some restrictions such that

$$Q(\alpha_i, y_i) = 0, 1 \leq i \leq n$$

- **Step2**: Factorize $Q(X, Y)$ and output $P(X)$ if

  1. $Y - P(X)$ is a factor of $Q(X, Y)$
  2. $deg(P) \leq k$
  3. $P(\alpha_i) = y_i$ for at least t values if i.

Last lecture, we saw an instantiation of the generic algorithm above, where we placed the following restrictions on $Q(X, Y)$: $deg_X(Q) \leq \sqrt{nk}$, $deg_Y(Q) \leq \sqrt{\frac{n}{k}}$. The algorithm was able to work with agreement $t \geq 2\sqrt{nk}$. In particular, rate R Reed-Solomon codes can be list decoded from $1 - 2\sqrt{R}$ fraction of errors. This bound is better than the unique decoding bound of $\frac{1-R}{2}$ for $R < 0.07$. This is still far from the $1 - \sqrt{R}$ fraction of errors guaranteed by the Johnson bound.

2 Algorithm 2

We now look at the list decoding algorithm in the breakthrough work of Sudan [2]. To motivate the algorithm, recall that in the previous algorithm, in order to prove that **Step 2** works, we defined a polynomial $R(X) \stackrel{\triangle}{=} Q(X, P(X))$. In particular, this implied that $deg(R) \leq deg_X(Q) + k \cdot deg_Y(Q)$ (and we had to set $t > deg_X(Q) + k \cdot deg_Y(Q)$). One shortcoming of this approach is that the maximum degree of X and Y might not occur at the same term. Sudan's insight was to use a more "balanced" notion of degree of $Q(X, Y)$:

Definition 2.1. *The $(1, k)$ weighted degree of the monomial X^iY^j is $i + kj$. Further, the $(1, k)$-weighted degree of $Q(X, Y)$ (or just its $(1, k)$ degree) is the maximum $(1, k)$ weighted degree of its monomials.*

Note that a bivariate polynomial $Q(X,Y)$ of $(1,k)$ degree at most D can be represented as follows:

$$Q(X,Y) \stackrel{\triangle}{=} \sum_{\substack{i+kj \leq D \\ i,j \geq 0}} q_{i,j} X^i Y^j.$$

Sudan's algorithm is basically the same as the list decoding algorithm we saw last lecture, except in the interpolation step, we compute a bivariate polynomial of bounded $(1,k)$ degree. More precisely, the algorithm is as follows:

- **Step 1:** Compute a non-zero polynomial $Q(X,Y)$ with $(1,k)$-weighted degree at most D (for some parameter D to be fixed later), such that for every $1 \leq i \leq n$:

$$Q(\alpha_i, y_i) = 0.$$

- **Step2**: Factorize $Q(X,Y)$ and output $P(X)$ if

  1. $Y - P(X)$ is a factor of $Q(X,Y)$
  2. $deg(P) \leq k$
  3. $P(\alpha_i) = y_i$ for at least t values if i.

As before to prove the correctness of the algorithm we need the following:

- **Step 1;** We will need to ensure that the number of coefficients of $Q(X,Y)$ is strictly greater than n.
- **Step 2:** Let $R(X) \stackrel{\triangle}{=} Q(X, P(X))$. We want to show that if $P(\alpha_i) \geq y_i$ for at least t values of i, then $R(X) \equiv 0$.

To begin with we argue why we can prove the correctness of **Step 2**. Note that by the definition of $(1,k)$ degree, $deg(R) \leq D$. Thus, using the same argument as we used for the list decoding algorithm from last lecture, we can ensure $R(X) \equiv 0$ if we pick $t > D$. Thus, we would like to pick D to be as small as possible. However, **Step 1** will need D to be large enough. Towards that end, let the number of coefficient of $Q(X,Y)$ be

$$N = |\{(i,j) | i + kj \leq D, i, j \in \mathbb{Z}^+\}|$$

To bound N, we first note that in the definition above, $j \leq \lfloor \frac{D}{k} \rfloor$. (For notational convenience, define $\ell = \lfloor \frac{D}{k} \rfloor$.) Consider the following sequence of relationships

$$\begin{aligned} N &= \sum_{j=1}^{\ell} \sum_{i=0}^{D-kj} 1 \\ &= \sum_{j=0}^{\ell} (D - kj + 1) \end{aligned}$$

$$
\begin{aligned}
&= \sum_{j=0}^{\ell}(D+1) - k\sum_{j=0}^{\ell} j \\
&= (D+1)(\ell+1) - \frac{k\ell(\ell+1)}{2} \\
&= \frac{\ell+1}{2}\left(2D+2-k\ell\right) \\
&\geq \left(\frac{\ell+1}{2}\right)(D+2) \qquad (1) \\
&\geq \frac{D(D+2)}{2k}. \qquad (2)
\end{aligned}
$$

In the above, (1) follows from the fact that $\ell \leq \frac{D}{k}$ and (2) follows from the fact that $\frac{D}{k} - 1 \leq \ell$.

Thus, **Step 1** is fine if $\frac{D(D+2)}{2k} > n$. The choice $D = \left\lceil \sqrt{2kn} \right\rceil$ suffices by the following argument:

$$\frac{D(D+2)}{2k} > \frac{D^2}{2k} \geq \frac{2kn}{2k} = n. \qquad (3)$$

Thus for **Step 2** to work, we need $t > \left\lceil \sqrt{2kn} \right\rceil$, which implies the following result:

Theorem 2.2. *Algorithm 2 can list decode Reed-Solomon codes of rate R from up to $1 - \sqrt{2R}$ fraction of errors. Further, the algorithm runs in polynomial time.*

Algorithm 2 runs in polynomial time as **Step 1** can be implemented using Gaussian elimination (and the fact that the number of coefficients is $O(n)$) while **Step 2** can be implemented by any polynomial time algorithm to factorize bivariate polynomials. Further, we note that $1 - \sqrt{2R}$ beats the unique decoding bound of $(1-R)/2$ for $R < 1/3$.

3 Algorithm 3

Finally, we present the list decoding algorithm for RS codes due to Guruswami and Sudan [1], which can correct $1 - \sqrt{R}$ fraction of errors. The main new idea is to add more restrictions on $Q(X, Y)$ (in addition to its $(1, k)$-degree being at most D). This change will have the following implications:

- The number of constraints will increase but the number of coefficients remains the same. This seems bad as this will result in an increase in D (which in turn would result in an increase in t).
- However, this change also increases the number of roots of $R(X)$ and this gain in the number of roots more than compensates for the increase in D.

In particular, the constraint will be as follows. For some integer parameter $r \geq 1$, we will insist on $Q(X, Y)$ having r roots at $(\alpha_i, y_i), 1 \leq i \leq n$.

To motivate the definition of multiplicity of a root of a bivariate polynomial, let us consider the following simplified example:

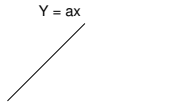

Fig 1.

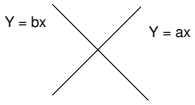

Fig 2.

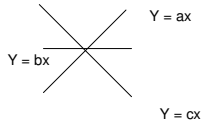

Fig 3.

In Fig. 1 the curve $Q(X, Y) = Y - aX$ passes through the origin once and has no term of degree 0. In Fig. 2, the curve $Q(X, Y) = (Y - aX)(Y - bX)$ passes though the origin twice and has no term of degree at most 1. In Fig. 3, the curve $Q(X, Y) = (Y - aX)(Y - bX)(Y - cX)$ passes through the origin thrice and has no term of degree at most 2. More generally, if r lines pass through the origin, then note that the curve corresponding to their product has no term of degree at most $r - 1$. This leads to the following more general definition:

Definition 3.1. *$Q(X, Y)$ has r roots at $(0, 0)$ if $Q(X, Y)$ doesn't have any monomial of degree at most $r - 1$.*

The definition of a root with multiplicity at a more general follows from a simple translation:

Definition 3.2. *$Q(X, Y)$ has r roots at (α, β) if $Q_{\alpha,\beta}(X, Y) \stackrel{\triangle}{=} Q(x + \alpha, y + \beta)$ have r roots at $(0, 0)$.*

We are now ready to state the formal algorithm:

- **Step 1:** Compute a non-zero polynomial $Q(X, Y)$ with $(1, k)$-weighted degree at most D (for some parameter D to be fixed later), such that for every $1 \leq i \leq n$:

$$Q(\alpha_i, y_i) = 0 \text{ with multiplicity } r.$$

- **Step2**: Factorize $Q(X, Y)$ and output $P(X)$ if

  1. $Y - P(X)$ is a factor of $Q(X, Y)$
  2. $deg(P) \leq k$

3. $P(\alpha_i) = y_i$ for at least t values if i.

To prove the correctness of the algorithm, we will need the following two lemmas:

Lemma 3.3. ***Step 1*** *implies* $\binom{r+1}{2}$ *constraints for each* i *on the coefficient of* $Q(X, Y)$.

Lemma 3.4. $R(X) \stackrel{\triangle}{=} Q(X, P(X))$ *has* r *roots at every* i *such that* $P(\alpha_i) = y_i$. *In other words,* $(X - \alpha_i)^r$ *divides* $R(X)$.

Using arguments similar to those used for Sudan's algorithm, we will need

$$\frac{D(D+2)}{2k} > n\binom{r+1}{2},$$

where the LHS is the number of coefficients of $Q(X, Y)$ as before and the RHS follows from Lemma 3.3. We note that the choice $D = \left\lceil \sqrt{knr(r-1)} \right\rceil$ works. Thus, we have shown the correctness of **Step 1**. For **Step 2**, we need to show that the number of roots of $R(X)$ (which by Lemma 3.4 is at least rt) is strictly bigger than the degree of $R(X)$, which as in Sudan's algorithm is still D. That is,

$$tr > D,$$

which is the same as

$$t > \frac{D}{r},$$

which in turn will follow if we pick

$$t = \left\lceil \sqrt{kn\left(1 - \frac{1}{r}\right)} \right\rceil.$$

If we pick $r = 2kn$, then we will need

$$t > \left\lceil \sqrt{kn - \frac{1}{2}} \right\rceil > \left\lceil \sqrt{kn} \right\rceil,$$

where the last inequality is because of t is an integer. Thus, we have shown

Theorem 3.5. *Algorithm* 3 *can list decode Reed-Solomon codes of rate* R *from up to* $1 - \sqrt{R}$ *fraction of errors. Further, the algorithm runs in polynomial time.*

The claim on the running time follows from the same argument that was used to argue the polynomial running time of Algorithm 2.

In the next lecture, we will prove Lemmas 3.3 and 3.4.

References

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

[2] M. Sudan. Decoding of Reed Solomon codes beyond the error-correction bound. *J. Complexity*, 13(1):180–193, 1997.