Implementing the Interpolation Step in the Guruswami-Sudan Algorithm

Peter Trifonov
Saint-Petersburg State Polytechnic University
Russia
Email: petert@dcn.ftk.spbstu.ru

Abstract

The interpolation step of the Guruswami-Sudan list decoding algorithm for Reed-Solomon codes is considered. Two improvements to the ideal exponentiation interpolation method are proposed.

I. Introduction

Reed-Solomon codes represent the most widely used family of error correcting codes. List decoding enables one to correct more errors compared to classical hard-decision decoding methods. However, the Guruswami-Sudan list decoding algorithm [1] still remains too computationally expensive to be used in practical systems. The most difficult part of this algorithm is construction of a bivariate polynomial having a number of roots of sufficiently high multiplicity.

A generalization of the binary exponentiation method to the case of zero-dimensional ideals was proposed in [2]. It consists of the multiplication stage, where bivariate polynomials having roots of multiplicity r are multiplied to double their root multiplicity, and the reduction stage, where linear transformations are applied to these polynomials to minimize their weighted degree. In this paper some modifications are proposed in order to reduce the complexity of the multiplication stage of this algorithm.

The paper is organized as follows. Section II presents an overview of the ideal exponentiation interpolation algorithm. Section III introduces some novel complexity reducing techniques. Numeric results are given in Section IV. Finally, conclusions are drawn.

II. Ideal exponentiation interpolation algorithm

A. Guruswami-Sudan list decoding algorithm

$(n, k, n-k+1)$ Reed-Solomon code over $GF(2^m)$ is defined as

$$RS(n,k) = \{(f(x_0), \dots, f(x_{n-1})) | f(x) \in GF(2^m)[x], \deg f(x) < k\},$$

where x_i are distinct elements of $GF(2^m)$, and n is usually set to $n = 2^m - 1$. List decoding of vector $Y = (y_0, \dots, y_{n-1})$ consists in finding all polynomials $f(x) : \deg f(x) < k$, such that $f(x_i) = y_i$ for at least τ positions i. This problem can be solved by the Guruswami-Sudan algorithm, which consists of the following stages:

1) Interpolaton: construct polynomial $Q(x, y)$ such that its $(1, k-1)$-weighted degree does not exceed l, and the points (x_i, y_i) are its roots of multiplicity r. By abuse of notation, we denote the latter propery as $Q(x_i, y_i) = 0^r$.
2) Factorization: find all polynomials $f(x) : Q(x, f(x)) = 0$ and $\deg f(x) < k$. Select among them those corresponding to valid solutions of the list decoding problem.

The expressions relating parameters n, k, r, l and τ can be found in [3].

The interpolation step can be implemented by solving a system of linear equations, but the dimension of this system causes the complexity of such implementation to be prohibitively high. A simpler way is to construct a Groebner basis of the ideal of interpolation polynomials.

B. Finding a Groebner basis of the ideal of interpolation polynomials

It can be easily shown that all polynomials $Q(x,y) : Q(x_i,y_i) = 0^r$ belong to ideal

$$I_r =< \phi^j(x)(y-T(x))^{r-j}, j=0..r >, \tag{1}$$

where $\phi(x) = \prod_{i=0}^{n-1}(x-x_i)$ and $T(x) : T(x_i) = y_i, i = 0..n-1$ [4], [5]. The required interpolation polynomial can be obtained by constructing a Groebner basis of this ideal with respect to $(1,k-1)$-weighted degree lexicographic monomial ordering, and selecting the smallest element of this basis. However, applying the generic Buchberger algorithm to polynomials given by (1) turns out to be a computationally difficult task. It is possible to reduce the complexity by constructing first a Groebner basis of I_1, and using it to obtain Groebner bases of ideals

$$I_{\sum_{j=l}^m r_j 2^{j-l}} = I^2_{\sum_{j=l+1}^m r_j 2^{j-l-1}} \cdot I_1^{r_l}, l = 0..m-1, \tag{2}$$

where $r = \sum_{j=0}^m r_j 2^j, r_j \in \{0,1\}$, $I^2 = I \cdot I$, $I^0 = GF(2^m)[x,y]$, and $I \cdot J$ denotes the product of ideals I and J. It is possible to show [6] that

$$< P_0,\dots,P_u > \cdot < S_0,\dots,S_v > =< P_i S_j, i = 0..u, j = 0..v >, \tag{3}$$

i.e. the product of ideals is generated by all pairwise products of their basis elements. Unfortunately, the basis of the ideal product obtained in this way is not a Groebner one even if the ideals being multiplied are given by their Groebner bases. Furthermore, the size of the Groebner basis of the ideal product is usually much smaller than $(u+1)(v+1)$, as given by (3). That is, most of the polynomial multiplication operations required by the standard ideal multiplication rule are useless, and computationally expensive Buchberger algorithm is needed to obtain a Groebner basis of the ideal product.

These problems were addressed in [2]. It was shown that the ideal product can be constructed by means of the $Merge$ algorithm, which is shown in Figure 2. It requires $O(u+v)$ bivariate polynomial multiplications. Furthermore, the Buchberger algorithm can be substituted with a generalization of the classical Euclidean algorithm presented in Figure 1. Here $rand()$ denotes a function returning a random non-zero element of $GF(2^m)$, $\operatorname{LT} Q(x,y)$ is the leading term of $Q(x,y)$ with respect to $(1,k-1)$-weighted degree lexicographic ordering, and $\operatorname{ydeg} Q(x,y) = j$ iff $\operatorname{LT} Q(x,y) = ax^u y^j$ for some $a \in GF(2^m)$ and $u \in \mathbb{Z}$. The bivariate interpolation algorithm based on this approach is shown in Figure 3. This algorithm constructs a Groebner basis of $I_r = \{Q(x,y) \in GF(2^m)[x,y] | Q(x_i,y_i) = 0^r\}$. The interpolation polynomial needed by the Guruswami-Sudan algorithm is given by the smallest element of this basis. Correctness proof and complexity analysis of this method are given in [2].

This approach reduces the interpolation complexity by the order of magnitude compared to the iterative interpolation algorithm [3]. However, it still remains quite high compared to classical Reed-Solomon decoding algorithms.

```
REDUCE$((S_0(x,y),\dots,S_{i-1}(x,y)),P(x,y))$
 1  $S_i(x,y) \leftarrow P(x,y)$
 2  while $\exists j : (0 \le j < i) \wedge (\mathrm{ydeg}\, S_j(x,y) = \mathrm{ydeg}\, S_i(x,y))$
 3  do if $\mathrm{LT}\, S_i(x,y) | \mathrm{LT}\, S_j(x,y)$
 4        then $W(x,y) \leftarrow S_j(x,y) - \frac{\mathrm{LT}\, S_j(x,y)}{\mathrm{LT}\, S_i(x,y)} S_i(x,y)$
 5             $S_j(x,y) \leftarrow S_i(x,y)$
 6             $S_i(x,y) \leftarrow W(x,y)$
 7        else $S_i(x,y) \leftarrow S_i(x,y) - \frac{\mathrm{LT}\, S_i(x,y)}{\mathrm{LT}\, S_j(x,y)} S_j(x,y)$
 8  if $S_i(x,y) = 0$
 9    then $i \leftarrow i - 1$
10  return $(S_0(x,y),\dots,S_i(x,y))$
```

Fig. 1. Construction of Groebner basis of zero-dimensional ideal $< S_0(x,y),\dots,S_{i-1}(x,y),P(x,y) >$.

```
MERGE$((P_0(x,y),\dots,P_u(x,y)),(S_0(x,y),\dots,S_v(x,y)),n,r_1,r_2)$
 1  $r \leftarrow r_1 + r_2$
 2  for $i \leftarrow 0$ to $u+v$
 3  do $Q_i(x,y) = \min_{0 \le j \le v} P_{i-j}(x,y) S_j(x,y)$
 4  $\mathcal{B} = (Q_0(x,y),\dots,Q_{u+v}(x,y))$
 5  while $\Delta(\mathcal{B}) > \frac{nr(r+1)}{2}$
 6  do $\alpha_i \leftarrow rand(), i = 0..u$
 7     $\beta_j \leftarrow rand(), j = 0..v$
 8     $Q(x,y) \leftarrow (\sum_{i=0}^{u} \alpha_i P_i(x,y)) (\sum_{i=0}^{v} \beta_i S_i(x,y))$
 9     $\mathcal{B} \leftarrow Reduce(\mathcal{B}, Q(x,y))$
10  return $\mathcal{B}$
```

Fig. 2. Construction of I_r basis from the Groebner bases of $I_{r_1} =< P_0(x,y),\dots,P_u(x,y) >$ and $I_{r_2} =< S_0(x,y),\dots,S_v(x,y) >$.

III. FAST IDEAL MULTIPLICATION

The main idea behind line 3 of $Merge$ algorithm is to find sufficiently small elements in $I_{r_1} I_{r_2}$, so that the number of iterations in the $WHILE$ loop is minimized. Furthermore, the constraint $\mathrm{ydeg}\, Q_i(x,y) = i$ is enforced. Most of the computation is performed on line 11 of $Interpolate$ algorithm, where the Groebner basis of $I^2 = I \cdot I$ is computed. At this step $Merge$ algorithm takes as input two identical vectors of bivariate polynomials. We propose to reduce the complexity by replacing lines 2 and 3 of $Merge$ algorithm with

$$Q_{2i}(x,y) = P_i^2(x,y) \tag{4}$$

$$Q_{2i+1}(x,y) = \min_{0 \le j \le u} P_{2i+1-j}(x,y) P_j(x,y), i = 0..u \tag{5}$$

This modification must be used only if identical ideals are multiplied. Observe that in the considered case of codes over $GF(2^m)$ one can use the identity $(a+b)^2 = a^2 + b^2$ to compute $P_i^2(x,y)$ with just L operations, where L is the number of non-zero terms in $P_i(x,y)$, while the straightforward polynomial multiplication requires $O(L^2)$ arithmetic operations. Numerical experiments indicate that this modification has almost no impact on the number of iterations of the subsequent $WHILE$ loop.

Furthermore, we observe that the bivariate polynomials $P(x,y) = \sum_i p_i(x) y^i$ arising in the considered algorithm have the "triangular" shape, i.e. $\deg p_i(x) \le c - (k-1)i$,

```
INTERPOLATE($((x_i, y_i), i = 1..n), r$)
 1  $\phi(x) \leftarrow \prod_{i=1}^{n}(x - x_i)$
 2  $T(x) \leftarrow \sum_{i=1}^{n} y_i \frac{\prod_{j\neq i}(x-x_j)}{\prod_{j\neq i}(x_i-x_j)}$
 3  $\mathcal{G} \leftarrow (\phi(x))$
 4  $j = 0$
 5  repeat
 6          $\mathcal{G} \leftarrow Reduce(\mathcal{G}, y^j(y - T(x)))$
 7          $j \leftarrow j + 1$
 8    until $\mathrm{LT}\, G_j = y^j$
 9  $\mathcal{B} \leftarrow \mathcal{G}$
10   Let $r = \sum_{j=0}^{m} r_j 2^j, r_j \in \{0, 1\}$
11  for $j \leftarrow m - 1$ to $0$
12  do $\mathcal{B} \leftarrow Merge(\mathcal{B}, \mathcal{B})$
13      if $r_j = 1$
14         then $\mathcal{B} \leftarrow Merge(\mathcal{B}, \mathcal{G})$
15  return $\mathcal{B}$
```

Fig. 3. Construction of a Groebner basis for I_r

for some $c > 0$. Bivariate polynomial multiplication, which is extensively used in $Merge$ algorithm, can be considered as two-dimensional linear convolution, and one can recursively use classical fast multiplication algorithms to compute it [7], [8]. However, the triangular shape of the polynomials would be completely destroyed during the pre-summation stage of these algorithms. Therefore we propose a modification of the Karatsuba algorithm adapted to the case of triangular polynomials.

Consider the problem of computing $C(x, y) = c_{00} + c_{01}y + c_{10}x + c_{11}xy + c_{02}y^2 + c_{20}x^2 = (a_{00} + a_{01}y + a_{10}x)(b_{00} + b_{01}y + b_{10}x)$. Degree-2 part of $C(x, y)$ can be computed as $y^2\left(a_{01} + a_{10}\frac{x}{y}\right)\left(b_{01} + b_{10}\frac{x}{y}\right) = y^2a_{01}b_{01} + xy\left((a_{01} + a_{10})(b_{01} + b_{10}) - a_{01}b_{01} - a_{10}b_{10}\right) + x^2a_{10}b_{10}$. The remaining part $C'(x, y) = c_{00} + c_{01}y + c_{10}x$ can be reconstructed from its values in any three points. An easy choice is $C'(0, 0), C'(0, 1)$, and $C'(1, 0)$. This leads to

$$\begin{aligned} C(x, y) = a_{00}b_{00} + \; & y\left((a_{00} + a_{01})(b_{00} + b_{01}) - a_{01}b_{01} - a_{00}b_{00}\right) + \\ & x\left((a_{00} + a_{10})(b_{00} + b_{10}) - a_{10}b_{10} - a_{00}b_{00}\right) + \\ & y^2a_{01}b_{01} + xy\left((a_{01} + a_{10})(b_{01} + b_{10}) - a_{01}b_{01} - a_{10}b_{10}\right) + x^2a_{10}b_{10} \end{aligned}$$

This algorithm requires 6 multiplications and 12 additions. Employing the conventional Karatsuba algorithm also requires 6 multiplications, but 14 additions. The coefficients a_{ij} and b_{ij} can also be triangular polynomials, enabling thus recursive application of the proposed algorithm. Indeed, one can decompose a generic triangular polynomial as

$$A(x, y) = \sum_{j=0}^{u-1}\sum_{i=0}^{(v-j)s-1} a_{ij}x^iy^j = \underbrace{\left(\sum_{j=0}^{u/2-1}\sum_{i=0}^{vs/2-1} a_{ij}x^iy^j\right)}_{A_{00}(x,y)} + X\underbrace{\left(\sum_{j=0}^{u/2-1}\sum_{i=0}^{(v/2-j)s-1} a_{vs/2+i,j}x^iy^j\right)}_{A_{10}(x,y)} + Y\underbrace{\left(\sum_{j=0}^{u/2-1}\sum_{i=0}^{(v/2-j)s-1} a_{i,u/2+j}x^iy^j\right)}_{A_{01}(x,y)},$$

TABLE I
LIST DECODING TIME, S

n, k, t, r	1	2	3
$31, 15, 10, 21$	0.49	0.46	0.45
$255, 200, 29, 17$	6.11	5.06	4.93

where$X = x^{vs/2}$, $Y = y^{u/2}$, and s is any integer. The polynomials $A_{00}(x, y)$, $A_{00}(x, y) + A_{10}(x, y)$ and $A_{00}(x, y) + A_{01}(x, y)$ are "rectangular", while $A_{01}(x, y) + A_{10}(x, y)$, $A_{10}(x, y)$ and $A_{01}(x, y)$ are again triangular. One can employ the above described algorithm recursively to compute the product of two such polynomials.

IV. Numeric results

Computer simulations were used to analyze the efficiency of the proposed improvements. Three versions of $Merge$ function were implemented:

1) The original version given in Figure 2.
2) Polynomial squaring method given by (4).
3) Polynomial squaring method with fast triangular polynomial multiplication.

Standard Karatsuba bivariate multiplication algorithm was used in the first two cases. The algorithms were implemented in C++ programming language, and simulations were performed on Intel Core i7 PC @2.67 GHz for different values of code length n, dimension k, number of errors t and root multiplicity r. The results are given in Table I. It can be seen that the polynomial squaring method provides up to 17 % better performance compared to the original version of $Merge$ algorithm. Additional 3 % gain can be achieved by using the proposed triangular polynomial multiplication method. Observe that the squaring method can be used jointly with re-encoding, which was shown to provide additional complexity reduction, as well as requires much less memory [2].

V. Conclusions

In this paper implementation issues of the ideal exponentiation interpolation algorithm were considered. It was shown that one can reduce the complexity of the multiplication stage of this algorithm by replacing polynomial multiplication operation with the squaring one. Furthermore, a modificaton of the Karatsuba algorithm was proposed for computing the product of "triangular" polynomials, which arise in the considered interpolation algorithm.

References

[1] V. Guruswami and M. Sudan, "Improved decoding of Reed-Solomon and algebraic-geometric codes," *IEEE Transactions on Information Theory*, vol. 45, no. 6, pp. 1757–1767, September 1999.

[2] P. Trifonov, "Efficient interpolation in the Guruswami-Sudan algorithm," *IEEE Transactions on Information Theory, submitted for publication*, December 2008.

[3] R. R. Nielsen and T. Hoholdt, "Decoding Reed-Solomon codes beyond half the minimum distance," in *Proceedings of the International Conference on Coding Theory and Cryptography*. Mexico: Springer-Verlag, 1998.

[4] K. Lee and M. E. O'Sullivan, "List decoding of Reed-Solomon codes from a Gröbner basis perspective," *Journal of Symbolic Computation*, vol. 43, no. 9, September 2008.

[5] P. Trifonov, "On the relationship of some Reed-Solomon decoding algorithms," in *Proceedings of "Coding Theory Days in Saint-Petersburg" Workshop*, 2008, pp. 83–87.

[6] D. Cox, G. Little, and D. O'Shea, *Ideals, varieties and algorithms*. Springer-Verlag, 1992.

[7] R. Blahut, *Theory and Practice of Error Control Codes*. Addison-Wesley, 1984.

[8] ——, *Fast Algorithms for Digital Signal Processing*. Addison-Wesley, 1985.