Improved hybrid algorithm for finding roots of error-locator polynomials

SERGEI FEDORENKO, PETER TRIFONOV,
St. Petersburg State Polytechnical University, Russia, *sfedorenko@ieee.org, petert@dcn.nord.nw.ru*

ELENA COSTA, HARALD HAAS
Siemens AG, *Elena.Costa@icn.siemens.de, Harald-1.Haas@icn.siemens.de*

October 23, 2002

Abstract. In this paper we suggest a hybrid method for finding toots of error locator polynomials. We first review a fast version of the Chien search algorithm based on the decomposition of the error locator polynomial into a sum of multiples of affine polynomials. We then propose to combine it with modified analytical methods for solution of polynomials of small degree in radicals.

We suggest, in particular, two efficient decompositions, whose combination with analytical algorithms yields a significant reduction in time-complexity, as proved by means of simulation.

Keywords: Chien search, error locator polynomial, p-polynomial, linearized polynomial, affine polynomial, BCH code, Reed-Solomon code

1 INTRODUCTION

It is well known that one of the most time-consuming stages of decoding process of Reed-Solomon, BCH and some other error-correcting codes is finding roots of the error-locator polynomial. Root finding problem can be formally stated as finding all distinct $x_i : F(x_i) = 0,\ F(x) = \sum_{j=0}^{t} f_j x^j,\ x_i, f_j \in GF(2^m)$. Many algorithms [1, 2, 3] for its solution were suggested, but for practical parameter values (currently most standards define Reed-Solomon codes with $t \leq 16,\ m = 8$) the most frequently used algorithm is the Chien search [7]. It is simple substitution of all elements of $GF(2^m)$ into the polynomial, it has very high time complexity.

Recently a modification of the Chien search algorithm based on a special polynomial evaluation rule has been suggested [5]. It is based on the decomposition of the error-locator polynomial into a sum of multiples of affine polynomials. In [5] it is also stated that the algorithm can be further improved by constructing different polynomial decompositions.

In this paper we propose two improved polynomial decompositions which may reduce implementation complexity of the root-finding algorithm for some error-correcting codes. We suggest a combination of the Chien search algorithm with a modification of analytical method.

The remainder of the paper is organised as follows. In section 2 we provide some basic definitions and describe the main idea behind the affine decomposition of polynomials. Later we present two special affine decompositions for polynomials frequently appearing in decoders of some popular error-correcting codes. In section 3 we describe some analytical techniques for finding roots of small-degree polynomials. Section 4 presents simulation results.

2 IMPROVED CHIEN SEARCH ALGORITHM

2.1 BASIC DEFINITIONS

In this section we introduce some basic definitions and describe the main idea of the affine decomposition as suggested in [5].

Definition 1. A polynomial $L(y)$ over $GF(2^m)$ is called a p-polynomial for $p = 2$ if

$$L(y) = \sum_i L_i y^{2^i},\ L_i \in GF(2^m).$$

These polynomials are also called linearized polynomials. The following lemma describes the main property of p-polynomials [1].

Lemma 1. *Let $y \in GF(2^m)$ and let $\alpha^0, ..., \alpha^{m-1}$ be a standard basis. If*

$$y = \sum_{k=0}^{m-1} y_k \alpha^k,\ y_k \in GF(2)$$

and $L(y) = \sum_j L_j y^{2^j}$, then

$$L(y) = \sum_{k=0}^{m-1} y_k L(\alpha^k).$$

Definition 2. A polynomial $A(y)$ over $GF(2^m)$ is called an affine polynomial if

$$A(y) = L(y) + \beta,\ \beta \in GF(2^m),$$

where $L(y)$ is a p-polynomial.

The above lemma makes possible evaluation of affine polynomials $A(x)$ with just one addition at each $x_i \in GF(2^m)$ if all x_i are ordered in their vector representation as Gray code.

Definition 3. Gray code is an ordering of all binary vectors of length m such that only one bit changes from one entry to the next.

So if $x_i \in GF(2^m)$ are ordered as a Gray code (i.e. $wt(x_i - x_{i-1}) = 1$, where $wt(a)$ is the Hamming weight of a) the following holds:

$$A(x_i) = A(x_{i-1}) + L(\Delta_i),\ \Delta_i = x_i - x_{i-1} = \alpha^{\delta(x_i, x_{i-1})}, \tag{1}$$

where $\delta(x_i, x_{i-1})$ indicates position in which x_i differs from x_{i-1} in its vector representation. If $x_0 = 0$ then $A(x_0) = \beta$ and the above equation describes the algorithm for evaluation of $A(x)$ at all points of $GF(2^m)$.

Example 1. Let us consider the case of $GF(2^3)$ defined by the primitive polynomial $\pi(\alpha) = \alpha^3 + \alpha + 1$. One of many possible Gray codes is the sequence 000, 001, 011, 010, 110, 111, 101, 100 or $0, 1, \alpha^3, \alpha, \alpha^4, \alpha^5, \alpha^6, \alpha^2$. So one needs to prepare a table of values $L(\alpha^0), L(\alpha^1), L(\alpha^2)$. Then $A(1) = A(0) + L(\alpha^0)$, $A(\alpha^3) = A(1) + L(\alpha^1)$ and so on.

This algorithm can be applied for evaluation of any polynomial if it is decomposed into a sum of affine multiples.

As stated in [5], every polynomial $F(x) = \sum_{i=0}^{t} f_i x^i$ can be decomposed into a sum of multiples of affine polynomials:

$$F(x) = f_3 x^3 + \sum_{i=0}^{\lceil (t-4)/5 \rceil} x^{5i} (f_{5i} + \sum_{j=0}^{3} f_{5i+2^j}\, x^{2^j}), \tag{2}$$

where $\lceil a \rceil$ is the smallest integer greater than or equal to a. Applying the polynomial evaluation algorithm described above to affine polynomials in (2) and using Horner's rule [8] for evaluation of the sum one can evaluate the polynomial at all non-zero points of the finite field $GF(2^m)$ with time complexity equal to

$$W_{fast} = m \left\lceil \frac{t+1}{5} \right\rceil (4C_{mul} + 3C_{add}) + (\left\lceil \frac{t+1}{5} \right\rceil (2C_{add} + C_{mul}) + 2C_{exp})(2^m - 1), \tag{3}$$

where $C_{exp}, C_{mul}, C_{add}$ denote the time complexity of one exponentiation, multiplication and addition over the finite field.

2.2 Some special decompositions

In this section we suggest decompositions of polynomials which can be used in decoders of Reed-Solomon $(255, 239, 17)$ and $(255, 223, 33)$ codes.

Example 2. A polynomial of degree not higher than 8 can be decomposed as

$$\begin{aligned} F(x) = &\ \textstyle\sum_{i=0}^{8} f_i x^i = & A_1(x) + x^3(A_2(x) + f_6 x^3), \\ A_1(x) = &\ f_0 + L_1(x) = & f_0 + f_1 x + f_2 x^2 + f_4 x^4 + f_8 x^8, \\ A_2(x) = &\ f_3 + L_2(x) = & f_3 + f_5 x^2 + f_7 x^4. \end{aligned} \tag{4}$$

The idea behind this example is to eliminate one exponentiation operation by moving the reminder term into an affine polynomial. The time complexity of evaluation of a polynomial at all points of $GF(2^m)$ using this decomposition equals to

$$C_8 = m(6C_{mul} + 4C_{add}) + (4C_{add} + 2C_{mul} + C_{exp})\,(2^m - 1). \tag{5}$$

Comparing this expression with (3) one can see that this decomposition reduces initialization cost and eliminates one exponentiation operation.

Example 3. The above example can be generalized to the case of polynomials of degree 17 as follows:

$$\begin{aligned} F(x) = &\ A_1'(x) + x^3(A_2(x) + x^3(f_6 + x^3(A_3(x) + x^3(A_4(x) + f_{15}x^3)))), \\ A_3(x) = &\ f_9 + f_{10}x + f_{11}x^2 + f_{13}x^4 + f_{17}x^8, \\ A_4(x) = &\ f_{12} + f_{14}x^2, \end{aligned} \tag{6}$$

where $A_1'(x) = A_1(x) + f_{16}x^{16}$. The time complexity of evaluation of a polynomial at all points of $GF(2^m)$ using this decomposition equals to

$$C_{17} = m(12C_{mul} + 8C_{add}) + (9C_{add} + 5C_{mul} + C_{exp})\,(2^m - 1). \tag{7}$$

One can see that this decomposition also reduces initialization cost with respect to (3) by eliminating one exponentiation operation, but it introduces one more addition and multiplication. Hence, one has to consider implementation costs of multiplication and exponentiation operations.

2.3 Combining Chien search with other methods

There exist already a lot of methods [2, 3, 4] for finding roots of polynomials of small degrees with very low complexity. Therefore, the combination of the Chien search with one of these techniques seems to have significantly lower complexity compared to the full Chien search. This can be easily implemented by division if the original error locator polynomial $F(x)$ by a factor $(x - x_i)$, after the root x_i has been discovered. Special low-degree algorithm can be invoked immediately when degree of the polynomial becomes sufficiently small.

In this section we investigate statistical behaviour of the Chien search. Let us suppose that the Chien search is applied to a polynomial $F(x)$ having t roots in the set $X = \{x_i \in GF(2^m)\backslash 0 | i = 1..n\}$, $n = 2^m - 1$. The probability that the k-th element of X will be the d-th root found by using the Chien search is equal to

$$p\{F(x_k) = 0_d\} = \frac{\binom{k}{d}\binom{n-k}{t-d}}{\binom{n}{t}} \frac{d}{k}.$$

This is the probability that the first k values chosen from X contain d roots of $F(x)$ and the last one also is a root.

So the average time complexity of finding d roots is

$$W = \sum_{k=1}^{n} C(t) k\, p\{F(x_k) = 0_d\} = C(t) \frac{d}{\binom{n}{t}} \sum_{k=1}^{n} \binom{k}{d}\binom{n-k}{t-d},$$

where $C(t)$ is the time complexity of one evaluation of the polynomial of degree t using some algorithm. Using well-known identity

$$\sum_{k=0}^{r} \binom{r-k}{m} \binom{s+k}{n} = \binom{r+s+1}{m+n+1}$$

we obtain

$$W = C(t) d \frac{n+1}{t+1}. \tag{8}$$

One can see that for polynomials with initially small t (that is the case of a good underlying channel) the application of a hybrid method can dramatically reduce the overall complexity.

3 Analytical methods for finding roots of polynomials of degree $t \leq 4$

In this section we describe some algorithms which can be used for finding roots of small-degree polynomials. The ideas behind these algorithms are widely known [1, 4, 6]. We present here some improvements.

3.1 Polynomials of degree 2

Since all polynomials of degree 2 are affine, one can find their roots solving the corresponding system of linear algebraic equations. However, for polynomials $P(x) = x^2 + x + a$ it is possible to compute all roots using just one matrix multiplication [6]. Equation $P(x) = 0$ can be represented in vector form as $\widehat{x}L = \widehat{a}$, where $\widehat{x} = (x_0, x_1, ..., x_{m-1})$, $\widehat{a} = (a_0, a_1, ..., a_{m-1})$ and

$$L = \begin{pmatrix} L(\alpha^0) \\ L(\alpha) \\ \cdots \\ L(\alpha^{m-1}) \end{pmatrix},$$

with $L(y) = y^2 + y$. Since $y^2 + y = 0$ only if $y = 0$ or $y = 1$, the rank of L is $m - 1$. Thus, there is a $m \times m$ matrix M such that

$$LM = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ & & & \cdots & & & \\ 0 & 0 & 0 & \cdots & 1 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 \end{pmatrix}$$

(in some fields the right side of this expression can differ from this by a permutation). Then solutions of the original equation (if they exist) can be represented as $\widehat{x} = (c, d_0, d_1, ..., d_{m-2})$, where $d = \widehat{a}M = (d_0, d_1, ..., d_{m-1})$, $c \in \{0, 1\}$. Solutions exist if and only if $d_{m-1} = 0$. Since matrix M can be computed for any field at design time finding roots of a polynomial can be performed using very simple logical scheme. Moreover, since there exist many values of M it is possible to choose one minimizing the implementation complexity of the scheme.

3.2 Polynomials of degree 3

Roots of any polynomial $P(x) = x^3 + ax^2 + bx + c$ can be expressed using Lagrange resolvents [4]

$$\begin{aligned} r_0 &= \quad x_0 + x_1 + x_2 \quad = a \\ r_1 &= x_0 + wx_1 + w^2x_2 \\ r_2 &= x_0 + w^2x_1 + wx_2, \end{aligned}$$

where $w^3 = 1$. Roots can be found by Inverse Fourier Transform of the resolvent vector:

$$\begin{aligned} x_0 &= \quad r_0 + r_1 + r_2 \\ x_1 &= r_0 + w^2r_1 + wr_2 \\ x_2 &= r_0 + wr_1 + w^2r_2. \end{aligned}$$

It is possible to show [6] that Lagrange resolvents satisfy $g(R_i) = 0$, where $g(x) = x^2 + (ab + c)x + (a^2 + b)^3$, $R_i = r_i^3, i = 1, 2$. Roots of this equation can be found using the algorithm described in section 3.1. However, since in some fields cubic root operation may be ambiguous it may be necessary to correct computed values of r_i using the Viete theorem [4].

However, computations can be simplified if the cubic polynomial $P(x)$ is multiplied by $(x + a)$ obtaining the polynomial of degree 4 $\widehat{P}(x) = x^4 + qx^2 + px + r$. This introduces additional root which should be removed from the output of the degree-4 solver.

3.3 Polynomials of degree 4

Roots $x_1, ..., x_4$ of a polynomial $P(x) = x^4 + ax^3 + bx^2 + cx + d$ can be found using values

$$\begin{aligned} \theta_1 &= (x_1 + x_2)(x_3 + x_4) \\ \theta_2 &= (x_1 + x_3)(x_2 + x_4) \\ \theta_3 &= (x_1 + x_4)(x_2 + x_3), \end{aligned}$$

which are roots of $q(x) = x^3 + (ac + b^2)x + (abc + a^2d + c^2) = 0$ [4, 6]. Then ξ_1, ξ_2 are roots of $\xi^2 + a\xi + \theta_1$, and η_1, η_2 are roots of $\eta^2 + (\theta_1 + b)\eta + d$. Then roots of the original polynomial are roots of $x^2 + \xi_1 x + \eta_1$ and $x^2 + \xi_2 x + \eta_2$.

If $a \neq 0$ computations can be simplified by substitution $x = \frac{1}{y} + \sqrt{\frac{c}{a}}$ (if $a = 0$ transformation is not required). Then the original polynomial transforms to $g(y) = y^4 + E_1 y^2 + E_2 y + E_3$, where

$$E_1 = \frac{\sqrt{ac} + b}{d + \frac{bc}{a} + \frac{c^2}{a^2}}, \quad E_2 = \frac{a}{d + \frac{bc}{a} + \frac{c^2}{a^2}}, \quad E_3 = \frac{1}{d + \frac{bc}{a} + \frac{c^2}{a^2}}.$$

Then $\theta_1 = r_0 + r_1 + r_2$, where r_i are Lagrange resolvents of equation $x^3 + E_1^2 x + E_2^2 = 0$. $r_0 = 0$, and $R_{1,2} = r_{1,2}^3$ are roots of

$$r^2 + E_2^2 r + E_1^6 = 0.$$

Then $r_i^3 = E_2^2 p_i$, p_i can be found using the algorithm for finding roots of $p^2 + p + p_0 = 0$, where

$$p_0 = \frac{E_1^6}{E_2^4} = \frac{(\sqrt{ac} + b)^6}{a^4(d + \frac{bc}{a} + \frac{c^2}{a^2})^2} = E_2^2 \left(\sqrt{c/a} + b/a\right)^6.$$

Since for some fields $r_{1,2}$ values can be not unique it may be necessary to take cubic root of only one value (e.g. r_1) and compute another as $r_2 = \frac{E_1^2}{r_1'}$. It can be seen that $\xi = \xi_1 = \xi_2 = \sqrt{\theta_1}$. Thus roots of the transformed polynomial are equal to roots of $y^2 + \xi y + \eta_1 = 0$ and $y^2 + \xi y + \eta_2 = 0$, where η_i are roots of $\eta^2 + (\theta_1 + E_1)\eta + E_3 = 0$.

4 Simulation results

To demonstrate the efficiency of the proposed solution, the hybrid algorithm has been implemented in C++ programming language and simulations have been performed. The multiplication of field elements in $GF(2^8)$ was implemented using tables of logarithms and antilogarithms. The computation times required to evaluate the polynomials at the field elements $\alpha^0, \ldots, \alpha^{254}$ were averaged over 100000 computations and shown in Table 1.

One can see that algorithms based on decomposition are significantly faster than the Chien search. Moreover, elimination of some computationally expensive operations dramatically reduces the overall complexity. And, as follows from (8), combination with analytical algorithms reduces the complexity even more.

5 Conclusions

In this paper we suggested improvements to some existing techniques which can significantly speedup one of the most time-consuming stages of the decoding process of Reed-Solomon, BCH and many other codes. Hybrid approach used in the paper has good potential and can lead to further algorithmic improvements.

Table 1: Computation time in microseconds for finding roots of polynomials

Degree	Chien search	General affine decomposition (2)	Special decomposition (4)	Hybrid method based on (4)	Special decomposition (6)	Hybrid method based on (6)
5	14.6	13.4	10.0	2.3	—	—
6	17.2	14.8	10.1	3.6	—	—
7	19.6	14.9	10.8	4.9	—	—
8	22.2	15.1	10.8	5.8	11.4	6.7
9	24.9	15.4	—	—	15.5	9.5
10	27.3	16.9	—	—	16.4	11.0
11	29.8	18.0	—	—	16.4	11.6
12	32.3	18.1	—	—	16.4	12.2
13	34.8	18.2	—	—	16.9	13.5
14	37.3	18.2	—	—	17.4	14.2
15	39.8	20.1	—	—	17.4	14.8
16	42.3	21.3	—	—	17.9	15.8
17	44.9	21.3	—	—	17.9	16.2

REFERENCES

[1] E.R. Berlekamp. *Algebraic coding theory*. New York: McGraw-Hill, 1968.

[2] C.-L. Chen. Formulas for the solutions of quadratic equations over $GF(2^m)$. *IEEE Transactions on Information Theory*, 28(5):792–794, 1982.

[3] R.T. Chien, B.D. Cunningham, and I.B. Oldham. Hybrid methods for finding roots of a polynomial with application to BCH decoding. *IEEE Transactions on Information Theory*, 15(2):329–335, 1969.

[4] B.L. Van der Waerden. *Algebra*, volume 1. Springer-Verlag, 1991.

[5] S.V. Fedorenko and P.V. Trifonov. Finding roots of polynomials over finite fields. *Accepted for publication in IEEE Transactions on Communications*, 2002.

[6] C.J. Williamson. Apparatus and method for error correction. U.S. Patent 5,905,740, May 1999.

[7] R.T. Chien. Cyclic decoding procedures for Bose-Chaudhuri-Hocquenghem codes. *IEEE Transactions on Information Theory*, 10(4):357–363, 1964.

[8] D.E. Knuth. *The art of computer programming*, volume 2. Addison-Wesley, 1998.