Transactions Letters

Finding Roots of Polynomials Over Finite Fields

Sergei V. Fedorenko, *Member, IEEE,* and Peter V. Trifonov, *Student Member, IEEE*

***Abstract*—In this letter, we propose an improved algorithm for finding roots of polynomials over finite fields. This makes possible significant speedup of the decoding process of Bose–Chaudhuri–Hocquenghem, Reed–Solomon, and some other error-correcting codes.**

***Index Terms*—Affine polynomial, Bose–Chaudhuri–Hocquenghem (BCH) code, Chien search, error-locator polynomial, linearized polynomial, p-polynomial, Reed–Solomon code.**

I. Introduction

IT IS WELL KNOWN that one of the most time-consuming stages of the decoding process of Reed–Solomon, Bose–Chaudhuri–Hocquenghem (BCH), and some other codes is finding roots of the error-locator polynomial. The most widely known root-finding algorithm is the Chien search method, which is a simple substitution of all elements of the field into the polynomial, so it has very high time complexity for the case of large fields and polynomials of high degree.

In [1], it was shown that every polynomial of degree not higher than five can be transformed into a canonical form with one or two parameters, so it is possible to construct tables for finding roots. Moreover, if some roots are located in the same cyclotomic coset, it is possible to eliminate them using the Euclidean algorithm. In their recent paper [2], Truong, Jeng, and Reed proposed a transformation which allows grouping of some summands of the polynomial of degree not higher than 11 into multiples of affine polynomials. Since affine polynomials can be easily evaluated using very small precomputed tables, it is possible to speed up computations. However, their algorithm suffers from some drawbacks:

1) it can be applied only to polynomials of degree not higher than 11;
2) transformation of the polynomial is required. Transformation proposed by authors ($y = x + f_6/f_7$ for polynomial $F(x) = \sum_{i=0}^{11} f_i x^i$) can not be applied if $f_7 = 0$, so the root-finding algorithm becomes more complicated;
3) after transformation the polynomial contains summand $f_{10}y^{10} + f_9 y^9$ (and $f_6 x^6$ if transformation failed). Evaluation of it still requires usage of Chien's algorithm.

Paper approved by V. K. Bhargava, the Editor for Coding and Communication Theory of the IEEE Communications Society. Manuscript received August 10, 2001; revised February 11, 2002. This work was supported by the Alexander von Humboldt Foundation, Germany.

The authors are with the Department of Distributed Computing and Networking, St. Petersburg State Polytechnic University, 194021, St. Petersburg, Russia (e-mail: sfedorenko@ieee.org).

Digital Object Identifier 10.1109/TCOMM.2002.805269

In this letter, we propose a generalized approach which can be used for decomposition and fast evaluation of any polynomial. We describe it for the case of $GF(2^m)$, but our results can be generalized for the case of an arbitrary field. This technique can be used in realization of Chien search.

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, and $f_j \in GF(2^m)$. Chien search algorithm solves it by evaluation of $F(x)$ at all $x \in GF(2^m)\backslash\mathbf{0}$ with the time complexity

$$W = (C_{\text{add}} + C_{\text{mul}})t(2^m - 1) \tag{1}$$

where C_{add} and C_{mul} are the time complexities of one addition and multiplication in the finite field, respectively. The algorithm described below reduces the cost of one polynomial evaluation using special reordering of field elements.

II. Fast Polynomial Evaluation Algorithm

Before description of the algorithm, let us first consider some definitions and properties.

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

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

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

Lemma 1 [3]: Let $y \in GF(2^m)$ and let $\alpha^0, \ldots, \alpha^{m-1}$ be a standard basis. If

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

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

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

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

0090-6778/02$17.00 © 2002 IEEE

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:

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

where $\delta(x_i, x_{i-1})$ indicates the 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, α^3, α, α^4, α^5, α^6, α^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.

Statement 1: Each polynomial $F(x) = \sum_{j=0}^{t} f_j x^j$, $f_j \in GF(2^m)$ can be represented as

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

where $\lceil a \rceil$ is the smallest integer greater than or equal to a.

Proof: Let k be the smallest integer, such that $5k - 1 \geq t$, and assume that for all $i > t$ $f_i = 0$. Then, the above equation can be represented as

$$\begin{aligned} F(x) &= F_k(x) \\ &= f_3 x^3 + \sum_{i=0}^{k-2} x^{5i} \left(f_{5i} + \sum_{j=0}^{3} f_{5i+2^j} x^{2^j} \right) \\ &\quad + x^{5(k-1)} \left(f_{5(k-1)} + \sum_{j=0}^{2} f_{5(k-1)+2^j} x^{2^j} \right). \end{aligned}$$

For $t = 4$ $(k = 1)$, this is obvious. Let us assume that $F_k(x)$ has been decomposed as described. Then, $F_{k+1}(x) = F_k(x) + x^{5k}(f_{5k} + f_{5k+1}x + f_{5k+2}x^2 + f_{5k+4}x^4) + x^{5(k-1)} f_{5(k-1)+8} x^8$. The last summand of this expression can be grouped with the last summand of the decomposition of $F_k(x)$. □

p-polynomials appearing in this decomposition have only four summands. In some cases, introducing additional summands can reduce the total amount of affine polynomials in the final decomposition.

So the whole root-finding algorithm is as follows:

1) compute $L_i^{(k)} = L_i(\alpha^k)$, $k = [0; m-1]$, $i \in [0; \lceil (t-4)/5 \rceil]$, where $L_i(x)$ are p-polynomials appearing in the above decomposition: $L_i(x) = \sum_{j=0}^{3} f_{5i+2^j} x^{2^j}$;
2) initialize $A_i^{(0)} = f_{5i}$;
3) represent each $x_j \in GF(2^m)$, $j \in [0; 2^m - 1]$ in standard basis as an element of Gray code with $x_0 = 0$, compute $A_i^{(j)} = A_i^{(j-1)} + L_i^{(\delta(x_j, x_{j-1}))}$, $j \in [1; 2^m - 1]$;
4) compute $F(x_j) = f_3 x_j^3 + \sum_{i=0}^{\lceil (t-4)/5 \rceil} x_j^{5i} A_i^{(j)}$, $j \in [1; 2^m - 1]$, and $F(0) = f_0$. If $F(x_j) = 0$, then x_j is a root of the polynomial. Note that the second summand of this sum can be computed using Horner's rule.

TABLE I
COMPUTATION TIME IN MICROSECONDS FOR EVALUATING THE POLYNOMIALS

Degree	Chien search	TJR method	New method	New method speedup rate
6	17.2	16.7	14.9	1.15
7	19.8	18.2	15.1	1.31
8	22.2	19.6	15.2	1.46
9	24.6	20.3	15.3	1.60
10	27.2	20.9	17.3	1.57
11	29.6	20.6	18.2	1.62
16	42.3	—	21.4	1.97
24	61.8	—	25.8	2.39
32	81.4	—	31.4	2.59

The total time complexity of this algorithm consists of complexity of preliminary computations (first summand) and complexity of polynomial evaluation, and is equal to

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

where C_{exp} denotes the time complexity of one exponentiation over the finite field.

III. Simulation Results

To demonstrate the efficiency of the new algorithm, it has been implemented in C++ programming language, compiled with MS Visual C++ 6.0 compiler, and software simulation on AMD Athlon 1700 XP processor on Windows XP operating system has 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 100 000 computations and shown in Table I.

Note that speedup rates for Truong, Jeng, and Reed's method are significantly lower than shown in [2]. This is caused by different implementation of the multiplication operation used in our simulations.

Comparing expressions (1) and (2) and corresponding experimental results, one can see that this algorithm can be up to 2.6 times faster than Chien search depending on implementation of operations over $GF(2^m)$.

IV. Conclusions

In this letter, we proposed an algorithm for evaluation of arbitrary polynomials at many points of the finite field with significantly better performance than the well-known Chien search.

Sometimes, performance of this algorithm can be further improved by construction of different polynomial decompositions.

ACKNOWLEDGMENT

The authors would like to thank Prof. V. K. Bhargava, Editor for Coding and Communication Theory, and the anonymous reviewers for their constructive comments.

REFERENCES

[1] R. T. Chien, B. D. Cunningham, and I. B. Oldham, "Hybrid methods for finding roots of a polynomial with application to BCH decoding," *IEEE Trans. Inform. Theory*, vol. IT–15, pp. 329–335, 1969.

[2] T.-K. Truong, J.-H. Jeng, and I. S. Reed, "Fast algorithm for computing the roots of error locator polynomials up to degree 11 in Reed–Solomon decoders," *IEEE Trans. Commun.*, vol. 49, pp. 779–783, May 2001.

[3] E. R. Berlekamp, *Algebraic Coding Theory*. New York: McGraw-Hill, 1968.