Computing 24, 51—60 (1980)

Computing

On the Complexity of Algorithms for the Translation of Polynomials

A. G. Akritas and **S. D. Danielopoulos,** Raleigh, N. C.

Received July 15, 1978

Abstract — Zusammenfassung

On the Complexity of Algorithms for the Translation of Polynomials. Three algorithms for computing the coefficients of translated polynomials are discussed and compared from the point of view of complexity. The two classical translation algorithms based on explicit application of the Taylor expansion theorem and the Ruffini-Horner method, respectively, have complexity $O(n^2)$. A third algorithm based on the fast Fourier transform is shown to have complexity $O(n \log n)$. However, when the cost of arithmetic operations is explicitly taken into consideration, the Ruffini-Horner algorithm is one order of magnitude better than the one based on the Taylor expansion and competes quite well with the algorithm based on the fast Fourier transform.

Über die Komplexität von Algorithmen zur Entwicklung von Polynomen. Wir vergleichen die Komplexität von 3 Algorithmen zur Berechnung der Polynomkoeffizienten an beliebigen Anschlußstellen. Die beiden klassischen Algorithmen beruhen auf der expliziten Anwendung der Taylor-Formel und der Methode von Ruffini-Horner. Beide haben Komplexität $O(n^2)$. Ein dritter Algorithmus verwendet die schnelle Fourier-Transformation und hat die Komplexität $O(n \log n)$. Wenn man aber die unterschiedlichen Kosten der verschiedenen arithmetischen Operationen genauer in Betracht zieht, dann ist der Algorithmus von Ruffini-Horner um eine Größenordnung besser als die Taylor-Entwicklung und mit dem Algorithmus, der auf schneller Fourier-Transformation beruht, durchaus vergleichbar.

1. Introduction

Algorithms which perform transformations of the form $x = x' + t$ on polynomials are of great interest because of certain new methods for the isolation of the roots of polynomials, which use such transformations [3]. For the sake of brevity, we shall call such a transformation a translation of the polynomial. The operation of interest in such a translation, is the computation of the coefficients of the translated polynomial. The design of efficient algorithms for the computation of these coefficients is very important to both numerical mathematics and computer algebra systems.

An obvious method of computing the coefficients of a translated polynomial is to use Taylor's expansion theorem, but the resulting algorithm is relatively inefficient. This is the approach used by Vincent [4], [9] in his remarkable root-isolation method. A better algorithm results from the use of the Ruffini-Horner method [5], [6], which in essence is a repeated application of the synthetic division

0010-485X/80/0024/0051/$ 02.00

algorithm and is used in the Akritas method for the isolation of real roots [3]. In recent years, the fast Fourier transform method has been used for the efficient evaluation of the convolution of two vectors [1], [2]. Since the coefficients of a translated polynomial can be expressed as a convolution of two vectors, the fast Fourier transform method provides the possibility for constructing better translation algorithms.

In the following sections we shall discuss briefly the two older methods which we shall call "classical" and we shall compare them to a method based on the fast Fourier transform, which to our knowledge has not yet been applied to algorithms requiring the translation of polynomials.

We shall compare these methods by obtaining, for each one of them, estimates for two closely related quantities that are proportional to the computation time of the corresponding algorithms. We shall define these quantities on the basis of a model of computation which, although oversimplified, will be adequate for comparing the methods considered here. We shall adopt the straight-line code model [2], according to which a computation is assumed to be performed by a sequence of assignment statements of the form $A \leftarrow B \,\omega\, C$, where A, B and C are variable names, constants, or subscripted variables, and ω is one of the operators $+$, $-$, $\times$ or $/$. A sequence of such statements will be called a straight-line algorithm, and the number of these statements will be called the complexity of the algorithm. When, for one reason or another, the execution of the arithmetic operations takes time which is comparable to the time needed for storing the results of the operation in memory, we shall assign to the operation a cost represented by the symbol $\mathscr{C}(A \,\omega\, B)$ and which will depend on the operation ω and the length (number of digits) of the operands. We shall define the costs for each one of the previously mentioned arithmetic operations in the following way:

ω	$\mathscr{C}(A \,\omega\, B)$
$+$ or $-$	$\max(l(A), l(B))$
$\times$ (or $*$)	$l(A) \cdot l(B)$
$/$	$(l(A) - l(B) + 1)\, l(B)$

In the above definitions, $l(A)$ represents the length of the operand A and is defined by the relation

$$l(A) = \begin{cases} \text{floor}(\log_b |A|) + 1, & A \neq 0 \\ 1, & A = 0 \end{cases}$$

where b indicates the base of the number system in which the operand A is represented when the operation ω is being performed. The notation $\mathscr{C}(A)$ will be used to denote the cost for computing the value of the entity A which may be either a scalar or a vector.

In this study we shall not consider the more sophisticated methods for performing multiplication [1]. Algorithms will be compared in terms of the cost for the classical multiplication method.

2. Algorithms for the Translation of Polynomials with the Classical Methods

Consider a polynomial $P(x)$ of degree $n-1$, $n>1$, with real coefficients $a_0, a_1, \ldots, a_{n-1}$, written in the form

$$P(x)=\sum_{i=0}^{n-1} a_i x^i. \tag{1}$$

Let t be a real value by which the polynomial is to be translated. According to Taylor's expansion theorem

$$P(x+t)=\sum_{k=0}^{n-1} \frac{P^{(k)}(t)}{k!} x^k=\sum_{k=0}^{n-1} c_k x^k. \tag{2}$$

Explicit evaluation of the derivatives $P^{(k)}(x)$ gives

$$P^{(k)}(x)=\sum_{i=k}^{n-1} i(i-1)\ldots(i-k+1)\, a_i x^{i-k}=k!\sum_{i=k}^{n-1} \binom{i}{k} a_i x^{i-k} \tag{3}$$

and, therefore, the coefficients c_k, $(k=0, 1, \ldots, n-1)$, of the translated polynomial are

$$c_k=\sum_{i=k}^{n-1} \binom{i}{k} a_i t^{i-k},\ (k=0, 1, \ldots, n-1). \tag{4}$$

The binomial coefficients $\binom{i}{k}$, which, from now on, we shall denote for simplicity by β_{ki}, can be obtained from Pascal's triangle [7]. The coefficients c_k, as given by formula (4), can be computed with the following scheme:

$$\begin{gathered} x_0 \leftarrow 1,\ \beta_{00} \leftarrow 1 \\ \beta_{0j} \leftarrow 1,\ x_j \leftarrow x_{j-1} * t,\ (j=1, 2, \ldots, n-1) \end{gathered} \tag{5}$$

$$\begin{aligned} &c_k \leftarrow a_k \\ &\beta_{k+1,j} \leftarrow \beta_{k+1,j-1}+\beta_{k,j-1} \\ &c_k \leftarrow c_k+\beta_{kj} * a_j * x_{j-k},\ (k=0, 1, \ldots, n-2;\ j=k+1, \ldots, n-1) \\ &c_{n-1} \leftarrow a_{n-1}. \end{aligned} \tag{6}$$

A straight-line algorithm can be obtained in an obvious manner from this scheme. It can be seen from (5) and (6) that the complexity of this algorithm is

$$\begin{aligned} T(n)&=\sum_{k=0}^{n-2} [4(n-1-k)+1]+2(n-1)+2 \\ &=2n^2+n-1. \end{aligned} \tag{7}$$

The Ruffini-Horner method avoids the computation of the binomial coefficients. It can be easily deduced by writing

$$\begin{aligned} &P(x)=(x-t)\, Q_{n-2}(x)+R_0 \\ &Q_{n-1-k}(x)=(x-t)\, Q_{n-2-k}(x)+R_k,\ (k=1, 2, \ldots, n-1) \end{aligned} \tag{8}$$

which, in fact, corresponds to performing repeatedly synthetic division. From these relations we obtain by substitution

$$P(x)=\sum_{k=0}^{n-1} R_k (x-t)^k \tag{9}$$

and

$$P^{(k)}(t)=k!\, R_k,\ R_k=Q_{n-1-k}(t) \tag{10}$$

where R_k are precisely equal to the coefficients c_k of the translated polynomial, and according to (8) and (10) are the remainders of the synthetic division of $Q_{n-1-k}(x)$, $(k=0, 1, \ldots, n-1)$ by $x-t$, where $Q_{n-1}(x)\equiv P(x)$. They are obtained with a straight-line algorithm corresponding to the computational scheme

$$\begin{aligned} &c_{j0}\leftarrow a_{n-1}, c_{0j+1}\leftarrow c_{0j} * t + a_{n-1-(j+1)}, (j=0, 1, \ldots, n-1)\\ &c_{kj}\leftarrow c_{kj-1} * t + c_{k-1,j}, (k=1, 2, \ldots, n-2; j=1, 2, \ldots, n-1-k)\end{aligned} \tag{11}$$

where $c_{k,n-1-k}\equiv c_k\equiv R_k$. The use of a (triangular) matrix (c_{kj}), instead of a vector, is necessary so that additional assignment statements that increase the complexity may be avoided. The complexity of this method can be deduced from (11) and is

$$\begin{aligned} T(n)&=3n+\sum_{k=1}^{n-1} 2(n-1-k)\\ &=n^2+2.\end{aligned} \tag{12}$$

Comparison of relations (7) and (12) show that both algorithms are of complexity $O(n^2)$ even though the Ruffini-Horner method is more elegant and about 2 times faster. In practice, however, the Ruffini-Horner method is much better. This can be seen from a computing time analysis of the previous two algorithm when a cost is assigned to arithmetic operations.

According to the computational model mentioned in Section 1, the cost for computing the vector of coefficients c_k, $(k=0, 1, \ldots, n-1)$, according to the algorithm implied by (5) and (6) is

$$\mathscr{C}(c)=\sum_{k=0}^{n-2}\sum_{j=k+1}^{n-1} [\mathscr{C}(\beta_{kj} * a_j)+\mathscr{C}(p_{kj} * x_j)+\mathscr{C}(r_{kj})] \tag{13}$$

where

$$p_{kj}=\beta_{kj} a_j,\ q_{kj}=p_{kj} x_j,\ r_{kj+1}=r_{kj}+q_{kj+1},\ (j=k+1, \ldots, n-1)$$

$r_{kk+1}=q_{kk+1}$, and x_j is defined as in (5). The computing time for r_{kj}, which is a sum, is proportional to the length of its largest term and, therefore, the cost for this operation is

$$\mathscr{C}(r_{kj})=l(r_{kj})=\max(l(r_{kj-1}), l(q_{kj})),\ (j=k+1, \ldots, n-1). \tag{14}$$

From a recursive use of this relation, it can be easily deduced that

$$\mathscr{C}(r_{kj})=l(q_{kj}). \tag{15}$$

The cost for computing the vector c is then

$$\mathscr{C}(c)=\sum_{k=0}^{n-2}\sum_{j=k+1}^{n-1} \{l(a_j)+[1+l(a_j)]\, l(x_j)+[1+l(a_j)+l(x_j)]\, l(\beta_{kj})\}. \tag{16}$$

In order to obtain a bound for this expression, we notice that

$$l(a_j) \leq \alpha \equiv l(\| \boldsymbol{a} \|_\infty), (j=0, 1, \ldots, n-1)$$

where $\underline{a}$ is the vector of coefficients of the given polynomial (1), and

$$l(x_j) < 1 + \tau j, (j=0, 1, \ldots, n-1),$$

where $\tau \equiv l(t)$ is the length of the translation parameter t. We have, therefore,

$$\begin{aligned} \mathscr{C}(\boldsymbol{c}) &< \tfrac{1}{2}(2\alpha+1)\, n(n-1) + \tfrac{1}{6}(\alpha+1)\, \tau\, n(n-1)(2n-1) \\ &+ (\alpha+2) \sum_{k=0}^{n-2} \sum_{j=k+1}^{n-1} l(\beta_{kj}) + \tau \sum_{k=0}^{n-2} \sum_{j=k+1}^{n-1} j\, l(\beta_{kj}). \end{aligned} \tag{17}$$

Estimates for the last two summations in (17) can be obtained by starting with the definition of the function $l(x)$ and the observation that

$$\begin{aligned} \sum_{j=k+1}^{n-1} l(\beta_{kj}) &< \operatorname{floor}\left(\sum_{j=k+1}^{n-1} \log \beta_{kj} \right) + (n-1-k) \\ &= \operatorname{floor}\left(\log \prod_{j=k+1}^{n-1} \beta_{kj} \right) + (n-1-k) \end{aligned} \tag{18}$$

and further noticing that

$$\log \prod_{j=k+1}^{n-1} \beta_{kj} = \log \prod_{j=1}^{n-1-k} \left(\frac{k+j}{j} \right)^{n-k-j} = \sum_{j=1}^{n-1-k} (n-k-j) \log \left(\frac{k+j}{j} \right). \tag{19}$$

Similar observations lead to the relations

$$\sum_{j=k+1}^{n-1} j\, l(\beta_{kj}) < \operatorname{floor}\left(\log \prod_{j=k+1}^{n-1} (\beta_{kj})^j \right) + n - 1 - k \tag{20}$$

and

$$\begin{aligned} \log \prod_{j=k+1}^{n-1} (\beta_{kj})^j &= \log \prod_{j=1}^{n-1-k} \left(\frac{k+j}{j} \right)^{S_{kj}} \\ &= \tfrac{1}{2} \sum_{j=1}^{n-1-k} [n(n-1) - (k+j)(k+j-1)] \log \left(\frac{k+j}{j} \right) \end{aligned} \tag{21}$$

where

$$S_{kj} = \sum_{m=k+j}^{n-1} m.$$

After the insertion of relations (18)—(21) into (17) and after using the fact that for a monotonically decreasing function we have

$$\int_1^{n+1} f(x)\, dx < \sum_{k=1}^{n} f(k) < \int_1^n f(x)\, dx + f(1) \tag{22}$$

we obtain

$$\mathscr{C}(\boldsymbol{c}) < \tfrac{1}{8} \tau n^4 + \tfrac{1}{3}(\alpha+1)\, \tau n^3 + \tfrac{1}{6}(\alpha+2) n^3 - \tfrac{1}{3} \tau n^3 \log n + O(\alpha n^2) \tag{23}$$

which we write as

$$\mathscr{C}(\boldsymbol{c}) = O(n^4 \log t + \alpha n^3 \log t). \tag{24}$$

It should be noted that in evaluating the cost for the computation of the coefficients $\boldsymbol{c}$, the costs for computing the binomial coefficients and the powers of t were not included in (16). These quantities were considered precomputed and stored. However, the inclusion of these costs in (16) would not have affected the order of magnitude in (23) or (24).

The cost of the Ruffini-Horner method, as described by the computational scheme (11), is given by

$$\begin{aligned}\mathscr{C}(\boldsymbol{c}) = &\sum_{j=0}^{n-2}[\mathscr{C}(c_{0j} * t) + C(p_{0j+1} + a_{j+1})] \\ &+ \sum_{k=1}^{n-2}\sum_{j=1}^{n-1-k}[\mathscr{C}(c_{kj-1} * t) + \mathscr{C}(p_{kj-1} + c_{k-1j})]\end{aligned} \tag{25}$$

where $p_{kj} = c_{kj} * t$, $(k=0, 1, \ldots, n-1;\ j=0, 1, \ldots, n-1-k)$. Using again the definitions of Section 1 and following procedures analogous to those used in the previous case, we obtain

$$\mathscr{C}(\boldsymbol{c}) < \tfrac{1}{6}\tau(\tau+1)(n-2)(n-1)n + \tfrac{1}{2}(\alpha\tau + \alpha + \tau)(n-1)n \tag{26}$$

or

$$\mathscr{C}(\boldsymbol{c}) = O\left(n^3 \log^2 t + n^2 \log t \log \frac{\| a \|_\alpha}{t}\right). \tag{27}$$

A comparison of costs (24) and (27) for the two algorithms indicates that in practice the Ruffini-Horner method can be almost one order of magnitude better than the explicit Taylor expansion method, provided that the translation parameter is not too large.

3. An Algorithm Based on the Fast Fourier Transform

The importance of the fast Fourier transform is due to the fact that it can be used for the computation of the convolution of two vectors in $O(n \log n)$ steps [1]. It is obvious that relation (3) can be rewritten so that after the substitution $x = t$, we obtain

$$P^{(k)}(t) = \sum_{i=k}^{n-1} i(i-1)\ldots 1\, a_i \frac{t^{i-k}}{(i-k)!} = \sum_{i=0}^{n-1} i!\, a_i \frac{t^{i-k}\,\theta(i-k)}{(i-k)!} \quad (k=0, 1, \ldots, n-1) \tag{28}$$

where we have defined

$$\theta(p) = \begin{cases} 1 \text{ for } p \geq 0 \\ 0 \text{ for } p < 0. \end{cases} \tag{29}$$

After a further renaming of the summation subscript, (28) can be written

$$P^{(k)}(t) = \sum_{i=0}^{n-1} \beta_i \alpha_{k-i}, \ (k=0, 1, \ldots, n-1) \tag{30}$$

where

$$\beta_i \equiv \frac{t^i}{i!},\ \alpha_i \equiv (n-1-i)!\, a_{n-1-i}\,\theta(i). \tag{31}$$

Clearly, each sum in (30) represents one element of the convolution of the vectors $\boldsymbol{\beta}=(\beta_0, \beta_1, \ldots, \beta_{n-1})^T$ and $\boldsymbol{\alpha}=(\alpha_0, \alpha_1, \ldots, \alpha_{n-1})^T$ [1] with elements defined by (31). For two arbitrary vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, of equal size n, we shall denote by $\boldsymbol{u} \circledast \boldsymbol{v}$ their convolution vector, and by $\boldsymbol{u} \circ \boldsymbol{v}$ the vector

$$\boldsymbol{u} \circ \boldsymbol{v}=(u_0 v_0, u_1 v_1, \ldots, u_{n-1} v_{n-1})^T. \tag{32}$$

Let

$$\boldsymbol{f}_n=\left(1, \frac{1}{1!}, \frac{1}{2!}, \ldots, \frac{1}{(n-1)!}\right). \tag{33}$$

Then the coefficient-vector of the translated polynomial $P(x+t)$ as defined in (2) is given by

$$\boldsymbol{c}=(\boldsymbol{\beta} \circledast \boldsymbol{\alpha}) \circ \mathbf{f}_n \tag{34}$$

where the elements of the vectors α and β are defined in (30). From relation (30), it can be easily seen that the complexity of the straight-line code computation of the convolution in (34) is $O(n^2)$. This convolution, however, can be computed in only $O(n \log n)$ steps with the fast Fourier transform algorithm [1]. For a review of this method, we refer the reader to the literature (see for example [8] and [1]). Here we shall present in outline form the corresponding computational scheme; this will be sufficient for our discussion of the complexity of the translation of polynomials.

The relationship between polynomials and the discrete Fourier transform can be easily deduced by comparing formula (1) with the formula

$$\tilde{a}_k=\sum_{j=0}^{n-1} a_j w^{kj}, \ (k=0, 1, \ldots, n-1) \tag{35}$$

which gives the k-th component of the discrete Fourier transform of the vector $\boldsymbol{a}=(a_0, a_1, \ldots, a_{n-1})^T$. Here

$$w=e^{-i\frac{2\pi}{n}}, \ i=\sqrt{-1}. \tag{36}$$

The corresponding inverse transform is obtained with the formula

$$a_k=\tfrac{1}{n}\sum_{j=0}^{n-1} \tilde{a}_j w^{-kj}, \ (k=0, 1, \ldots, n-1). \tag{37}$$

We shall use the notation $F(\boldsymbol{a})$ to represent the discrete Fourier transform vector of vector $\boldsymbol{a}$ as it is computed with the fast Fourier transform algorithm, and $F^{-1}(\tilde{\boldsymbol{a}})$ to represent the corresponding inverse transform. It can be easily seen that the Fourier transform of the convolution of two vectors is equal to the pairwise product, defined in (32), of the transforms of the two vectors [1], i.e.,

$$F(\boldsymbol{u} \circledast \boldsymbol{v})=F(\boldsymbol{u}) \circ F(\boldsymbol{v}). \tag{38}$$

In view of this property, formula (34) that gives the coefficient-vector of the translated polynomial $P(x+t)$, can be written

$$\boldsymbol{c}=F^{-1}(F(\boldsymbol{\beta}) \circ F(\boldsymbol{\alpha})) \circ \mathbf{f}_n. \tag{39}$$

The fast Fourier transform of an arbitrary vector of size n $(u_0, u_1, \ldots, u_{n-1})^T$ can be obtained with the following computational scheme, in which it is assumed that $n=2^k$. Moreover, the n input values are stored in the first half of a vector $\boldsymbol{u}$ of size $2n$.

$$\left.\begin{array}{l} S_j \leftarrow 0, (j=0, 1, \ldots, 2(n-1)) \\ i \leftarrow \text{floor}(j/2) \\ S_j \leftarrow S_i/2 \\ S_{j+1} \leftarrow S_j + n/2 \end{array}\right\} \quad (j=2, 4, \ldots, 2(n-1)) \tag{40}$$

$$\left.\begin{array}{l} p \leftarrow 0;\ q \leftarrow n \\ r \leftarrow 2^{l-1};\ m \leftarrow n/r \\ t \leftarrow p;\ p \leftarrow q;\ q \leftarrow t \\ p \leftarrow r-2 \\ \sigma \leftarrow 2*(i-1)*m \\ p \leftarrow p+2 \\ \left.\begin{array}{l} \mu \leftarrow p+\sigma+j \\ v \leftarrow q+\sigma+j \\ u_\mu \leftarrow u_v + u_{v+m} * w^{S_p} \\ u_{\mu+m} \leftarrow u_v + u_{v+m} * w^{S_{p+1}} \end{array}\right\} (j=0, 1, \ldots, m-1) \end{array}\right\} (i=1, 2, \ldots, r/2) \Bigg\} (l=2, 3, \ldots, k+1) \tag{41}$$

The results of the computation, i.e., the components of the discrete Fourier transform, are recovered from either the first half or the second half of vector $\boldsymbol{u}$ according to whether $k=\log_2 n$ is even or odd. It should also be noted that these components are not recovered in their correct order. The latter can be accomplished by relocating the component with subscript, say j, into the location with subscript $i=\text{revrs}(j)$. The function *revrs* produces the integer i from j, by reversing the bits in the binary representation of j [1]. The same computational scheme can be used for obtaining the inverse transform provided that negative powers of w are used and the resulting components are divided by n as specified by formula (37).

A straight-line algorithm can be easily obtained from the computational scheme (41) and its complexity can be easily seen to be

$$T(n)=\sum_{l=2}^{k+1}\left[6+\sum_{i=1}^{r/2}\left(5+\sum_{j=0}^{m-1}8\right)\right]=\tfrac{1}{2}n\log n+5n+6\log n-5 \tag{42}$$

which confirms the previous statement that the discrete Fourier transform can be computed in $O(n \log n)$ steps. The computation of the coefficients of the translated polynomial of (2) requires three executions of the fast Fourier transform algorithm. The complete computation requires the computation of the vectors $\boldsymbol{\beta}$ and $\boldsymbol{\alpha}$, as defined in (31), and the application of formula (39) which requires two pairwise products of the form (32), two applications of the fast Fourier operator F and one application of the inverse F^{-1}. The resulting complexity is

$$T(n)=\tfrac{3}{2}n\log n+18n+18\log n-15$$

which, of course, is still $O(n \log n)$. When a cost is assigned to arithmetic operations, as in the case of the algorithms of Section 2, the costs for each of the three fast Fourier transforms required in (39), as deduced from (41), are

$$\begin{aligned}\mathscr{C}(F(\boldsymbol{\alpha})) &< (\omega+1)n^2 \log^2 n + (\omega+1)\alpha n \log n + (2\omega+1)n \log n - (\omega+1)n^2 \log n \\ &= O(\omega n^2 \log^2 n)\end{aligned} \tag{43}$$

$$\begin{aligned}\mathscr{C}(F(\boldsymbol{\beta})) &< \omega(\omega+1)n(n+1)\log n + (\omega+1)(n-1)\log n \log \tfrac{t}{n}\theta(t-n) + \omega n \log n \\ &= O(\omega^2 n^2 \log n + \omega n^2 \log n \log \tfrac{t}{n}\theta(t-n))\end{aligned} \tag{44}$$

$$\begin{aligned}\mathscr{C}(F^{-1}(\boldsymbol{\gamma})) &< (\omega+1)n^2 \log n \theta(n-t-1) + (\omega+1)n^2 \log n \log t \theta(t-n) + O(\alpha\omega n \log n) \\ &= O(\omega n^2 \log n \theta(n-t) + \omega n^2 \log n \log t \theta(t-n))\end{aligned} \tag{45}$$

Here

$$\boldsymbol{\gamma} = F(\boldsymbol{\alpha}) \circ F(\boldsymbol{\beta}) \tag{46}$$

and $\omega = l(\| \boldsymbol{w} \|_\infty)$, $\boldsymbol{w} = (1, w, w^2, \ldots . w^{n-1})^T$. Combining these three relations, we obtain

$$\mathscr{C}(F^{-1}(F(\boldsymbol{\alpha}) \circ F(\boldsymbol{\beta}))) = O(\omega n^2 \log^2 n \theta(n-t) + \omega n^2 \log n \log t \theta(t-n) + \omega^2 n^2 \log n) \tag{47}$$

which, when compared to (27), shows that the fast Fourier transform method is about one order of magnitude faster than the Ruffini-Horner method, especially when the magnitude of the translation parameter t is at least as large as the degree n of the polynomial. It should be noted that the validity of these conclusions is dependent on the manner in which arithmetic is performed. The costs for computing the vectors $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, and $\boldsymbol{\gamma}$ can be easily determined to be

$$\mathscr{C}(\boldsymbol{\alpha}) = O(\alpha n^2 \log n) \tag{47 a}$$

$$\mathscr{C}(\boldsymbol{\beta}) = O(n^2 \log n \theta(n-t) + n^2 \log t \log \tfrac{t}{n}\theta(t-n)) \tag{47 b}$$

$$\mathscr{C}(\boldsymbol{\gamma}) = O(n^3 \log n + n^3 \log n \log \tfrac{t}{n}\theta(t-n)). \tag{47 c}$$

We notice that the cost for computing the vector $\boldsymbol{\gamma}$ is higher than for computing the Fourier transforms themselves. This is due to the lengths of the elements of the vectors $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ which according to relations (28)—(31) must be constructed in a way that would allow the derivatives of the polynomial to be expressed as the convolution of these vectors. In a system in which all arithmetic is performed by the computer hardware, the fast Fourier method is clearly superior to the older methods. In a system, however, in which extended-precision rational arithmetic is required and the arithmetic operations are performed in part by software, the Ruffini-Horner method may still be the simplest and most efficient.

Acknowledgements

It is a pleasure to thank the referee for his constructive criticism and comments.

References

[1] Aho, A. V., Hopcroft, J. E., Ullman, J. D.: The design and analysis of computer algorithms. Reading-Menlo Park-London-Amsterdam: Addison-Wesley 1974.

[2] Aho, A. V., Steiglitz, K., Ullman, J. D.: Evaluating polynomials at fixed sets of points. SIAM J. Comput. *4*, 533—539 (1975).

[3] Akritas, A. G.: A new method for polynomial real root isolation. Proc. 16th Annual Southeast Regional ACM Conference, Atlanta, April 13—15, 39—43 (1978).

[4] Akritas, A. G., Danielopoulos, S. D.: On the forgotten theorem of Mr. Vincent. Historia Mathematica *5*, 427—435 (1978).

[5] Cajori, F.: Horner's method of approximation anticipated by Ruffini. Amer. Mathem. Soc. Bull. *17*, 409—414 (1911).

[6] Horner, W. G.: On algebraic transformation. The Mathematician (London) *1*, 108—112, 136—142, 311—316 (1845).

[7] Knuth, D. E.: The art of computer programming, Vol. 1: Fundamental algorithms, 2nd ed. Reading-Menlo Park-London-Don Mills: Addison-Wesley 1973.

[8] Schwarz, H. R.: Elementare Darstellung der schnellen Fouriertransformation. Computing *18*, 107—116 (1977).

[9] Vincent: Sur la résolution des équations numériques. Journal de Mathématiques Pures et Appliquées *1*, 341—372 (1836).

A. G. Akritas, Ph. D. Candidate
Prof. Dr. S. D. Danielopoulos
Department of Computer Science
North Carolina State University
Raleigh, NC 27650, U.S.A.