ON THE COMPUTATION OF POWERS
OF A CLASS OF POLYNOMIALS

Ellis Horowitz

Sartaj Sahni

TR 72 - 143

August 1972

Department of Computer Science
Cornell University
Ithaca, New York 14850

1. Introduction

Recently there has been a great deal of interest in efficient algorithms for computing powers of a polynomial, see [3, 4,5,6, and 7]. Each author has chosen one or more computational models of polynomials within which he analyzes various algorithms. Most of the past work has concentrated on the dense model (where all possible terms occur) first because it yields upper bounds on computing times and secondly because its characteristics are easily defined and subsequently analyzed. A further reason for this emphasis on the dense model has been the inability to reasonably define the notion of a sparse polynomial. Thus a major outstanding question is whether the most efficient methods for dense polynomials are also good for sparse polynomials (whatever they are). In this paper a general class of polynomials is defined which includes both the dense case and a sensible definition of sparseness. Then various algorithms are analyzed and it is shown when efficient methods for the dense model are also good for sparse polynomials.

Let us first review more concretely the previously considered models for dense polynomials. Let $P(x_1,\ldots,x_r)$ be a polynomial in r variables such that (deg(P) in x_i+1) $\leq$ m. Then if P has m^r non-zero terms (all those that are possible) P is said to be dense. Further it is assumed that P^i has $(im)^r$ non-zero terms and thus P^i remains dense for every power, $1\leq i\leq n$. The coefficients of P can be either floating point numbers, elements from some Galois field, denoted GF(p) where p is often a single precision odd prime, or variable precision integers. The latter case is, of course implemented in software and is extremely important in algebraic manipulation systems where all the numbers must be maintained exactly, see e.g. [2]. However, as shown in [6] the assumption of multiple precision coefficients can be easily reduced to polynomials whose coefficients are in $GF(p_i)$ for some set of single precision primes p_i. Thus in the main part of this paper it will be sufficient to assume that the coefficients of polynomials are either single precision floating point numbers or elements of GF(p). In both cases the time for the arithmetic operations on these coefficients is bounded by a constant.

Within the confines of this dense polynomial model, [6] reviews the computing times for the major 3 methods: a) the binary method; b) iteration; and c) a new Homomorphism algorithm. In terms of the 3 relevant parameters: m(the maximum degree), n (the power to which the polynomial is to be raised) and r (the number of variables) we have the following asymptotic results.

<u>BINARY</u>	<u>ITERATION</u>	<u>HOMOMORPHISM</u>
$(mn)^{2r}$	$m^{2r}n^{r+1}$	$(mn)^{r+1}$

Therefore, at present the Homomorphism algorithm is asymptotically the best method to use. Moreover the constants associated with all 3 of the methods are small and in [6], Horowitz shows that when $m \geq 3$, for r=2 and r=3 the Homomorphism method is already better for $n \geq 2$. Thus it is both practically and asymptotically a useful method.

In [4], Gentleman considers a different computational model for polynomials. Let P initially be a sum of q terms with floating point or GF(p) coefficients. Then he requires that P^n

have $\binom{q-1+n}{q-1}$ terms for all positive integers n. In general this means that the number of terms of P^n grows as $O((q+n)^n/n!)$. Though explicitly his model is independent of the number of variables of P and of the degrees of any of the variables, there is an implicit relationship. That is if P belongs to the class up to power n and has r variables then each variable must have a minimal degree which is very large. For this model Gentleman has shown that computing P^n by iteration will almost always be faster than using the binary method.

In Section 2 we will define an entire class of computational models of polynomials. This class will include both sparse and dense subcases. Then we will see how the polynomial type defined by Gentleman relates to this class. In Section 3 several algorithms will be analyzed for computing powers of polynomials within this broad class. In Section 4 we will see how to construct an algorithm for polynomial powers which combines the best of several good methods and which will be most efficient given any representative polynomial.

2. The Model

Let $P(x_1,\ldots,x_r)$ be a polynomial in $r\geq 1$ variables whose coefficients are either floating point numbers or elements of GF(p). Let us assume that each variable occurs with maximum exponent m-1 so that P can have at most m^r possible terms. Then q may take on values between 1 and m and if q^r is small compared to m^r then P is initially sparse. These assumptions greatly simplify the subsequent analysis and we note that the resulting algorithms will allow for different degrees in each variable. Therefore the following results will apply to an entirely general polynomial.

Since we are raising P to a power another consideration is the way in which the terms of P^i grow as we compute P^n. If $P= cx_1^{e_1} \cdots x_r^{e_r}$ where c is non-zero and $e_i \geq 0$ for $1 \leq i \leq r$ then q=1 and P is sufficiently sparse so that P^i has only one term for all i. On the other hand if P is initially

$$P = \sum_{1 \leq i \leq m^r} x_1^{e_{i,1}} \cdots x_r^{e_{i,r}}$$

where the exponents $(e_{i,1},\ldots,e_{i,r})$ assume all possible values in the range $0 \leq e_{i,j} \leq m-1$ then P is completely dense. Moreover, the number of non-zero terms in P^i is $(im)^r$ and there is no cancellation, since all coefficients are initially +1. In order to formalize this idea we introduce the following notion of the growth rate of a polynomial.

Suppose P has q^r non-zero terms. Then the table below indicates the extent to which terms may cancel as we compute successive powers of P.

$$
\begin{array}{lcccccl}
\text{Polynomials} = & \underline{P}, & \underline{P^2}, & \underline{P^3}, & \underline{P^4},\ldots, & \underline{P^n} & \underline{\text{Growth Rate}}
\end{array}
$$

$$
\begin{array}{c}\text{number}\\ \text{of}\\ \text{terms}\end{array}
\left\{
\begin{array}{cccccl}
q^r, & q^r, & q^r, & q^r,\ldots, & q^r & \text{Constant} \\
q^r, & 2q^r, & 3q^r, & 4q^r,\ldots, & nq^r & \text{Linear} \\
q^r, & 2^2q^r, & 3^2q^r, & 4^2q^r,\ldots, & n^2q^r & \text{Quadratic} \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots\ldots \\
q^r, & 2^kq^r, & 3^kq^r, & 4^kq^r,\ldots, & n^kq^r & k^{th} \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots\ldots \\
q^r, & 2^rq^r, & 3^rq^r, & 4^rq^r,\ldots, & n^rq^r & \text{Exponential}
\end{array}
\right.
$$

Thus we are really considering $r+1$ different models of polynomial growth, some of which can be considered sparse and others dense. Note that if $q=m$ and $k=r$ then this is precisely the dense model for polynomials that has been dealt with previously. Also k can exceed r as a function of q and m and we shall see this is the case for the Gentleman polynomials.

Now let us see how Gentleman's polynomials relate to our definitions. If P has q terms then we get the following growth pattern:

P	P^2	P^3	,. . . . . ,	P^n
q	$\frac{q(q+1)}{2!}$	$\frac{q(q+1)(q+2)}{3!}$		$\frac{q(q+1)\cdots(q+n-1)}{n!}$
2,	3,	4,	. . . . ,	$n+1$
3,	6,	10,	. . . ,	$(n+1)(n+2)/2$
4,	10,	20,	. . . ,	$(n+1)(n+2)(n+3)/3!$

Suppose we look more closely at the case of $q=3$. Thus, for P in one variable it has the form

$$P = x^a + x^b + x^c$$

where we will ignore the coefficients and assume for the moment that they are all +1. Then

$$P^2 = x^{2a} + x^{a+b} + x^{a+c} + x^{2b} + x^{b+c} + x^{2c}$$

which has the required 6 terms as long as no 2 exponents are equal. Further we see that

$$P^3 = x^{3a} + x^{2a+b} + x^{2a+c} + x^{a+2b} + x^{a+b+c} + x^{a+2c} + x^{3b} + x^{2b+c} + x^{b+2c} + x^{3c}$$

giving 10 terms for P^3 as long as none of the exponents are the same. Therefore, we see that Gentleman's definition implies no cancellation whatsoever. Moreover we see that all possible sums of exponents must be distinct. Thus for example

$$P = x^9 + x^7 + x^4$$

will be a member for $n=2$ and 3, but not for $n=4$. Thus, for a given n, P may be a member of Gentleman's class, but for sufficiently larger n it will no longer be. In the univariate case we note that given q and n, P must have degree at least 2^n in order to qualify for Gentleman's class. As the number of variables increase, it will be necessary that no 2 vectors of exponents are identical. Thus his definition really does imply a certain minimal degree for P in the r variables.

3. Computing Time Analysis

Given $P(x_1,\ldots,x_r)$, $r \geq 1$ with q^r terms and a k^{th} order growth rate let us analyze the computing time for forming $P^n(x_1,\ldots,x_r)$ using various algorithms:

3.1 Iteration

Here we simply compute $P^2, P^3, \ldots, P^n$. Then, the computing time will be

$$\sum_{1 \leq i \leq n-1} (i^k q^r) q^r = q^{2r} \sum_{1 \leq i \leq n-1} i^k = O(q^{2r} n^{k+1})$$

We note that if $q=m$ and $k=r$, which is the case for the dense model, then the time for iteration is $O(m^{2r} n^{r+1})$ which is confirmed by Horowitz in [6], pp.9.

3.2 Binary

Here we use the binary expansion of n to dictate when we square our polynomials and when to additionally multiply by P. A brief description of this algorithm which computes $R=P^n$ is given as follows:

1) $R \leftarrow 1$; $Z \leftarrow P$;
2) $q \leftarrow \lfloor n/2 \rfloor$; $r \leftarrow n-2q$; $n \leftarrow q$; If $r=0$, go to (4);
3) $R \leftarrow R \cdot Z$;
4) If $n=0$ then return (R);

   else $(Z \leftarrow Z^2$; go to (2))

The major portion of the time is taken up in Step (4) which is executed at most $O(\log_2 n)$ times. Thus the total time for this method is

$$\sum_{1\le i\le \log_2 n} (2^{ik}q^r)^2 = q^{2r} \sum_{1\le i\le \log_2 n} (2^{2k})^i = O(q^{2r}n^{2k})$$

Checking this result with our previous knowledge of the dense case, $q=m$, $k=r$ we see that the binary method takes $O((mn)^{2r})$.

Let us stop and consider the relative merits of these 2 well-known algorithms. The ratio of Iteration/Binary is $q^{2r}n^{k+1}/q^{2r}n^{2k} = 1/n^{k-1}$ which shows us the following

a) when $k=0$, i.e. the number of terms of P^i is bounded by q^r, $1\le i\le n$ then the <u>binary</u> method is superior;

b) when $k=1$ and the growth rate is linear then both methods are equivalent

c) for $k\ge 2$ iteration is superior to the binary method by a factor of $1/n^{k-1}$.

Heindel, in [5] was the first to show part (c) for the case $k=r$ and $q=m$, i.e. for completely dense polynomials. Later on we will see that for $k=r$, $q=m$ it will always be more efficient to use methods other than iteration.

3.3 Binomial

Here we will compute P^n using the formula

$$P^n = (R+S)^n = \sum_{0\le i\le n} \binom{n}{i} R^i S^{n-i}$$

where we split P in half so that R and S both have $q^r/2$ terms. We can assume that the binomial coefficients $\binom{n}{i}$ are pre-stored and furthermore that the algorithm is recursively called to compute R^i, S^i for $0\le i\le n$. In order to analyze the computing time for this method we get the following 2 variable recurrence equation:

If $T_n(q^r)$ is the time needed to compute the n^{th} power of a polynomial with q^r terms, then

$$T_n(q^r)=2(T_2(q^r/2) + T_3(q^r/2) +\cdots+T_n(q^r/2)) + \sum_{0\le i\le n} (\frac{i^k q^r}{2})(n-i)^k \frac{q^r}{2}$$

$$=2(T_2(q^r/2) +\cdots+T_n(q^r/2)) + q^{2r}/4 \sum_{0\le i\le n} [i(n-i)]^k$$

$$=2T_2(q^r/2)+\cdots+T_n(q^r/2))+ (q^{2r}/4)(n^{2k+1})$$

Now suppose we define

$$a_2i = T_2(q^r/2^i)+\cdots+T_{n-1}(q^r/2^i),\ 1\le i\le \log_2 q^r$$

Then

$$T_n(q^r) = \left(\sum_{1\le i\le \log_2 q} 2^i a_2 i\right) + q^{2r}n^{2k+1} \ge O(q^{2r}n^{2k+1})$$

Thus we see that the Binomial method is worse than both the Binary and Iterative algorithms. The ratios at best are Iteration/Binomial = $1/n^k$ and Binary/Binomial = $1/n$. For $k\ge 1$ we see that the Binomial method is an extremely poor way of computing polynomial powers. Interestingly, though, for k=0 where the Binary method is superior to Iteration, we see that the Binomial method is as good as the Binary method.

This splitting of the polynomials in the Binomial method closely resembles the binary method and this is why the computing time is subsequently worse. We can reduce this time by changing the way we do the splitting. Instead, let R be a single term and S the remainder of P. Then computing R^i for $2 \leq i \leq n$ takes only n-1 operations. Using iteration to compute S^i takes

$$\sum_{1 \leq i \leq n-1} i^k (q^r-1)^2 = O(q^{2r} n^{k+1})$$

and this will bound the time for all the remaining steps. Thus, for polynomial powers the monomial splitting turns out to be more efficient than splitting into 2 equal parts. But clearly this method will always be bounded by the time for Iteration. The idea of monomial splitting was first proposed by Fateman in [3].

3.4 Homomorphism

Now let us reconsider the Homomorphism algorithm as given in [6]. For convenience the algorithm is restated here and computes $R(x_1,\ldots,x_r) = P^n(x_1,\ldots,x_r)$:

1) [Initial Case] If $r=0$, $R \leftarrow P^n$ using the binary method; return R;

2) [Initialize] $m_{r-1} \leftarrow \deg(P)$ in x_{r-1}; $C \leftarrow 0$; $D \leftarrow 1$; $b \leftarrow -1$; $m_r \leftarrow \deg(P)$ in

3) [Choose next point] $b \leftarrow b+1$;

4) [Evaluate] $\bar{P}(x_1,\ldots,x_{r-1}) \leftarrow P(x_1,\ldots,x_{r-1},b)$; If $\deg(P)$ in $x_{r-1} < m_{r-1}$ then go to (3);

5) [Recursion] $\bar{R} \leftarrow$ HOMOMORPHISM$(\bar{P},n)$;

6) [Interpolate] Compute the unique interpolating polynomial $C(x_1,\ldots,x_r)$ that interpolates the points $(b_i, C(x_1,\ldots,x_{r-1},b_i))$ $1 \le i \le k$ and $(b, R(x_1,\ldots,x_{r-1}$ where b_i are all the previous points of evaluation in step(4).

7) [Done?] $D(x_r) \leftarrow (x_r - b)D(x_r)$ If $\deg(D) \le m_r n$ then go to (3)

8) [end!] $R \leftarrow C$; end.

By way of explanation if P is simply a constant, then the binary method is used since it requires only $O(\log_2 n)$ single precision multiplications. Otherwise, consecutive integer points, b are chosen and P is evaluated sufficiently often for its main variable at those points. After the recursion is completed an iterative interpolation is performed which builds the final answer. An analysis of the computing times for the individual steps are given in [1] or [6] and they will be assumed here.

In [6], Horowitz shows that for completely dense polynomials, this algorithm has a computing time of $O((mn)^{r+1})$. The new assumptions here are that P has q^r terms and a k^{th} order growth rate. However, since m is the maximum degree in each variable we note that the degree growth rate is

$$m, 2m, 3m, 4m, \ldots, nm$$

for

$$P, P^2, P^3, P^4, \ldots, P^n$$

Steps(3)-(7) of the algorithm are governed by this maximum degree and not by the number of non-zero terms. If we are to produce a polynomial with potentially $(mn)^r$ terms we must interpolate at $(mn)^r+1$ points and there seems to be no way of modifying the interpolation algorithm to take care of zero coefficients. Therefore, the asymptotic computing time remains $O((mn)^{r+1})$ no matter how many terms initially exist or despite the growth rate.

3.5 Fourier Transforms

The use of the convolution property of Fourier transforms plus the ability to compute the transform efficiently offers another alternative algorithm. Given a sequence $(a_0,\ldots,a_{m-1})$ of length m, then its Fourier transformed sequence $(\hat{a}_0,\ldots,\hat{a}_{m-1})$ is defined by

$$\hat{a}_i = \sum_{0\le j\le m-1} a_j r^{ij} \qquad 0\le i\le m-1$$

where r is a principal m^{th} root of unity. The inverse transform is defined as

$$a_i = 1/m \sum_{0\le j\le m-1} \hat{a}_j r^{-ij} \qquad 0\le i\le m-1$$

The fast Fourier transform algorithm provides a way of computing the transform of a sequence of length m in $O(m \log m)$ operations. References to this algorithm can be found in [7], pp. 428. Let

$$P(x_r) = \sum_{0\le i\le m-1} a_i x_r^i \quad \text{and} \quad P^n(x_r) = \sum_{0\le i\le mn-1} c_i x_r^i$$

where $P(x_r)$ has only q non-zero coefficients, but maximum degree m-1 in x_r. Also, P^n has only $n^{k/r}q$ non-zero coefficient , but maximum degree mn-1. These coefficients a_i and c_i are polynomials in r-1 variables. Then one method which uses Fourier transforms proceeds as follows:

1) Compute the Fourier transform of the sequence $(a_0,\ldots,a_{m-1},0,\ldots,0)$ of length mn giving $(\hat{a}_0,\ldots,\hat{a}_{mn-1})$;

2) Compute the sequence $(\hat{c}_0,\ldots,\hat{c}_{mn-1})$ where $\hat{c}_i = \hat{a}_i^n$ by applying the algorithm recursively;

3) Compute the inverse Fourier transform of $(\hat{c}_0,\ldots,\hat{c}_{mn-1})$ giving $(c_0,\ldots,c_{mn-1})$.

The fact that this method works is proven by Pollard in [8]. The computing time is

$$T(r) = q^{r-1}mn(\log(mn)) + mnT(r-1) + (mn)^{r-1}(mn)\log mn:$$

Each multiplication in step (1) takes q^{r-1} steps since the a_i has q^{r-1} terms. Step(2) is recursively applied mn times. In Step (3), the $\hat{c}_i$ in general will have $(mn)^{r-1}$ terms. Solving the recurrence equation yields

$$T(r) = O(\ (mn)^r\log(mn)^r)$$

Thus the method of Fourier transforms cannot capitalize on a polynomial with a small growth rate but like the Homomorphism method remains bounded by the maximum occurring degree. A second difficulty is in choosing an appropriate value for r. In order to choose r easily such that r^n=1 all arithmetic is usually done modulo r^n-1 where r^n-1 is a prime. A consequence of this approach is that it complicates the ability of dealing with polynomials whose coefficients are initially from some other finite field, an important area of application.

In order to overcome this difficulty there are 2 possibilities as described by Pollard in [8]. One way is to choose r^n-1 large enough so as to bound all numbers a_i, $\hat{a}_i$, c_i, $\hat{c}_i$, which occur in the algorithm. The time to determine if r^n-1 is prime takes exponentially long and so these values must be precomputed. Moreover, given n an r may not necessarily exist such that r^n-1 is prime and n may have to be increased until a suitable r can be found. This increases the length of the sequence, which minimally has length (mn) and thus increases the computing time. A second approach avoids the necessity of determining a single r^n-1 by choosing a sufficient number of $p_i=r_i^n-1$ $1\le i\le k$. Thus, even though information is lost when doing arithmetic modulo p_i, the correct results are obtained by applying the Chinese Remainder algorithm to the residues. The time for this process now becomes

$$3k(mn)^r\log(mn)^r + k^2(mn)^r$$

where the first term is for 3k applications of the fast Fourier transform and the second term is for applying the Chinese Remainder algorithm to the k residues each of $(mn)^r$ numbers. Since k can become arbitrarily large as a function of n, the time for this algorithm can become considerably worse than the minimum. Therefore only under severely restrictive conditions does the Fourier transform algorithm really offer a practical and efficient method for computing polynomial powers.

3.6 Sums

This method is given by Knuth [7] for raising a power series to a power. Fateman [3] analyzes this method for the case of a dense multivariate polynomial.

If P is a polynomial in r variables then we may write P and P^n as

$$P = \sum_{i=0}^{m} a_i x_r^i \qquad \text{and } P^n = \sum_{i=0}^{mn} c_i x_r^i$$

where a_i and c_i are polynomials in r-1 variables.

Knuth [7] obtains the following recurrence relations for the c_i's: $c_0 = a_0^n$ and

$$* \qquad c_i = \frac{1}{a_0 i} \sum_{1 \le j \le \min(m,i)} ((n+1)j-i) a_j c_{i-j}$$

$$= \frac{1}{a_0 i} b_i \qquad , \; 1 \le i \le mn$$

Since a_0 is a polynomial in r-1 variables an expensive division is needed to determine each c_i. An analysis of this algorithm yields $O(q^{3r} n^{2kr})$ which is extremely poor. However, it is possible to avoid the multivariate divisions in (*) by replacing the multivariate coefficient a_0 in P by a single variable a_0^* (as suggested in [3]) getting a polynomial P' in the r+1 variables $(x_1, x_2, \cdots x_r, a_0^*)$. The c_i' are now computed using (*). The divisor is now a single variable and so the division by a_0^* just requires an exponent subtraction in b_i'.

Assume now that P is regular with q^r non-zero terms (i.e. each non-zero a_j has q^{r-1} non-zero terms) and has a growth rate k. Then P^n has $n^k q^r$ non-zero terms and each c_j in P^n or c_j' in P'^n except c_0' has $n^{k(r-1)/r}\ q^{r-1}$ terms.

Computing each c_i' requires at most

$$q(n^{k(r-1)/r}q^{r-1})(q^{r-1}) = n^{k(r-1)/r}q^{2r-1}$$

time as there are only q non-zero a_i'. Since there are only $(n^{k/r}q + 1)$ non-zero c_i' in P^n (because of the growth rate assumption) all the c_i' can be computed in $O(n^k q^{2r})$ time. However one must determine in advance which c_i's will be non-zero in order to achieve this bound. This can be done by examining the indices of the non-zero a_i's in P producing a sequence of indices i for which $c_i \neq 0$. This can often be done in $O(n^{k/r}q)$ time, so the total time for SUMS is unchanged.

To obtain P^n from P'^n we substitute the r-1-variate polynomial a_0 for a_0^*. Consider P'^n as a polynomial with a_0^* as the main variable such that

$$P'^n = \sum_{i=1}^{n-1} d_i\ a_0^{*i} + a_0^{*n} + d_n$$

Since $\quad P' = R + a_0^*$

$\quad$ and $P'^n = \sum \binom{n}{i} R^{n-i} a_0^{*i}$

we have that the number of terms in d_i is equal to the number of terms in $R^{n-i} = (n-i)^k\ q^r < n^k q^r$

Then, using Horners rule to compute $\displaystyle\sum_{i=1}^{n-1} d_i a_0^{*i} + a_0^{*n}$

with $a_0^* = a_0$ requires n multiplications by the q^{r-1} term polynomial

a_0. The total time needed for this is

$$n\,(n^k q^r)(q^{r-1}) = n^{k+1} q^{2r-1}$$

Hence the total time required by SUMS is

$$O(n^k q^{2r} + n^{k+1} q^{2r-1})$$

For r=1, however, straightforward evaluation of (*) requires only $n^k q^{2r}$ steps. Comparing this with Iteration we have

$$\frac{\text{Sums}}{\text{Iteration}} = \frac{n^k q^{2r} + n^{k+1} q^{2r-1}}{n^{k+1} q^{2r}} = \frac{1}{n} + \frac{1}{q}$$

which suggests that Sums is asymptotically better than Iteration for all growth rates k. We note that our analysis assumes that P is regular and that our procedure for detecting the non-zero c_i's does not generate any incorrect indices. In addition we note that the constant for this method becomes rather high because of the substitutions. In practice because of its conceptual simplicity we recommend the use of Iteration over SUMS.

4. Conclusion

Our results are summarized in the table below:

Method	Computing Time
BINOMIAL (equal splitting)	$q^{2r}n^{2k+1}$
BINARY	$q^{2r}n^{2k}$
BINOMIAL(monomial splitting)	$q^{2r}n^{k+1}$
ITERATION	$q^{2r}n^{k+1}$
SUMS	$q^{2r}n^{k} + q^{2r-1}n^{k+1}$, $r>1$
HOMOMORPHISM	$(mn)^{r+1}$
FOURIER TRANSFORM	$(mn)^{r}\log(mn)^{r}$

The first 5 methods have computing times which depend upon the actual number of non-zero terms, q^r, which occur in the original polynomial. Methods 6 and 7 depend upon the maximum number of terms which could possibly occur, $(mn)^r$, given that m is the largest degree which is present. Thus, for rather dense polynomials with a large growth rate we naturally expect the latter 2 methods to be superior to the other 5. But as the polynomials become more sparse and/or the growth rate much smaller there will come a time when Iteration will be less expensive than Homomorphism or Fourier Transform. Let us try to determine where this cut-off is and develop a rule for deciding, a priori, which method to use.

Case (1): P is initially completely dense and no terms cancel as we compute powers. Thus $q=m$, $k=r$ and we have in decreasing order:

ITERATION	$m^{2r}n^{r+1}$
SUMS	$\begin{cases} m^{2r}n^{r}+m^{2r-1}n^{r+1}, & r>1 \\ m^{2}n, & r=1 \end{cases}$
HOMOMORPHISM	$(mn)^{r+1}$
FOURIER TRANSFORM	$(mn)^{r}\log(mn)^{r}$

Thus, theoretically the Fourier Transform is the best method. However, because of the difficulties of implementing this method as outlined in Section 3.5 we recommend in practice the Homomorphism method. Another reason is that the times for Homomorphism versus Fourier Transform are relatively close and the constants especially small.

Case (2): P is initially completely sparse, that is having only a single term. This degenerate case reduces to raising the coefficient to the n^{th} power plus r multiplications to give the new exponents. For computing the n^{th} power of a single precision number the binary method is preferable and optimality questions have been discussed by Knuth in [7], Sec.4.6.3.

Let us now consider the two really practical methods, Iteration versus Homomorphism and see when to use each one.

$$\frac{\text{Iteration}}{\text{Homomorphism}} = \frac{q^{2r}n^{k+1}}{(mn)^{r+1}} = \frac{q^{2r}}{m^{r+1}n^{r-k}}$$

Hence, Iteration will be a better method to use when

$$\frac{q^{2r}}{m^{r+1}n^{r-k}} < 1$$

and worse to use when the ratio is greater than one. Therefore, we are looking for the break even point in the equation

$$q^r = \sqrt{m^{r+1} n^{r-k}} \qquad (4.1)$$

This gives a test which is a function of both the initial number of terms and the growth rate. But the growth rate concept is difficult to determine and moreover, just as a car accelerates and decelerates so too can a polynomial vary in the number of terms it produces as the power increases. Since the growth rate is too difficult to determine we instead search for some reasonable guidelines upon which to choose a method.

For a growth rate of $k \leq r$ we see by equation (4.1) that Iteration will be better as long as

$$q^r < \sqrt{m^{r+1}}$$

or roughly the number of terms of P must be less than the square root of the total possible number of terms.

In particular if m=100 and r=2, then q^r must be less than 1001 terms or 1/10 of the total possible number of terms. In another example if m=100 and r=10 then q^r must be less than 10^{11} out of a possible 10^{20} terms.

For a growth rate of k=0 which includes a great deal of cancellation, this case becomes much more favorable to Iteration. We have that

$$q^r \leq \sqrt{(mn)^r m}$$

or roughly that the number of terms of P must be only less than the square root of the total possible number of terms in P^n. Thus, if m=100 and r=2, q^r must be less than about 1000n before Iteration is better. Thus, if P has 1/10 the total number of possible terms or 1000 terms, then for all $n>1$ Iteration will be more efficient.

Summarizing then we obtain the following results from the model:

(1). If $q^r \geq \sqrt{m^{r+1} n^r}$, then the Homomorphism method will always be superior;

(2) If $q^r < \sqrt{m^{r+1}}$, then Iteration is the preferred method as long as the growth rate k is bounded by r;

(3) If q^r is within the range $\sqrt{m^{r+1}} \leq q^r \leq \sqrt{m^{r+1} n^r}$, then Iteration is better only if a lot of cancellation is suspected, i.e. the growth rate k is small.

Now suppose that we extend our definition of growth rate to include polynomials whose order is greater than r. Clearly, the maximum growth rate for any polynomial is $\leq (nm)^r$. Let P's growth rate be defined as

$$\rho (im)^r, \text{ where } \left(\frac{q}{im}\right)^r \leq \rho \leq 1.$$

For $\rho=1$, P has the maximum possible rate which it can only really achieve if $q=m$. Again we seek to know when Iteration is better than Homomorphism.

$$\frac{\text{Iteration}}{\text{Homomorphism}} = \frac{q^r \sum\limits_{1 \leq i \leq n-1} \rho (im)^r}{(mn)^{r+1}} = \frac{\rho (qm)^r n^{r+1}}{(mn)^{r+1}}$$

Thus Iteration will be better to use when $\rho (qm)^r < m^{r+1}$ or $q^r < \frac{m}{\rho}$. Hence for the maximum value of $\rho=1$ it follows that when $q^r<m$ Iteration will always be the better method to use. Hence we are led to the following, a priori rules for deciding upon which method to use.

Always Use Iteration	Probably Use Iteration	Use Iteration only If the Suspected Growth [illegible]	Use Homomorphism
m	$\sqrt{m^{r+1} n^{r-k}}$		$\sqrt{(mn)^r m}$

On the above axis if $q^r \leq m$ then we <u>always</u> use Iteration. If $m < q^r < \sqrt{m^{r+1}}$ then Iteration will be good, as long as the growth rate is less than or equal to r. If $\sqrt{m^{r+1}} < q^r \leq \sqrt{(mn)^r m}$ then the Homomorphism method is preferred unless one knows in advance that the order of growth will be substantially bounded. Finally, for $q^r > \sqrt{(mn)^r m}$, the Homomorphism method will <u>always</u> be the superior method to choose.

Now how best can we insert these criteria into a polynomial powers algorithm? Suppose we have $P(x_1,\ldots,x_r)$ where $m_i = \deg(P)$ in x_i, $1 \leq i \leq r$ and q_i = the number of terms of P in which x_i has a positive exponent. Then m_i and q_i can be computed initially in linear time. We now insert step (1.b) into the Homomorphism algorithm which is given in Section 3.4. If i indicates the level of recursion then step (1.b) will read

$$\text{(1.b)}\quad q \leftarrow q_1 \cdots q_{r-i+1}; \quad m \leftarrow m_1 \cdots m_{r-i+1} \cdot \min_{1 \leq j \leq r-i+1} \{m_j\};$$

$$\text{If } q < \sqrt{m} \text{ then Return } (R \leftarrow \text{Iteration } (P,n));$$

This insertion is particularly helpful in the following sense: in $P(x_1,\ldots,x_r)$ some of the x_i may be sparse while others are dense. As we evaluate and decrease the number of variables by one at each level of recursion, the test is individually applied. Thus, at every level the algorithm maximizes its chances of computing $\overline{P}(x_1,\ldots,x_i)^n$ in the most efficient manner.

Therefore, we have produced a rule which takes only linear time to compute and with which we can reasonably decide whether to use Iteration or Homomorphism. The resulting algorithm offers the best features of algorithms for manipulating both sparse and dense polynomials.

References

1) Collins, G.E., Heindel, L.E., Horowitz, E., McClellan M.T., Musser, D.R., "The SAC-1 Modular Arithmetic System", U. of Wisconsin Technical Report No. 10, Madison June 1969.

2) Collins, G.E. "The SAC-1 System:An Introduction and Survey" Proc. 2nd Symposium on Symbolic and Algebraic Manipulation SIGSAM , Los Angeles, March 1971.

3) Fateman, Richard J. "On the Computational Powers of Polynomials", Department of Mathematics Report, M.I.T. Cambridge, Mass.

4) Gentleman W.M., "Optimal Multiplication Chains for Computing a Power of a Symbolic Polynomial" SIGSAM Bulletin No.18, April 1971, (to appear in Mathematics of Computation).

5) Heindel, L.E.,"Computation of Powers of Multivariate Polynomials over the Integers", Journal of Computer and System Science, vol No.1, Feb, 1972.

6) Horowitz, E.,"The Efficient Calculation of Powers of Polynomials" Journal of Computer and System Science, (to appear).

7) Knuth, D.E., The Art of Computer Programming, Vol.2 "Semi-numerical Algorithms", Addison Wesley, 1969.

8) Pollard J.M., "The Fast Fourier Transform in a Finite Field", Mathematics of Computation, No. 114, April 1971.