Improved Techniques for Factoring Univariate Polynomials†

GEORGE E. COLLINS AND MARK J. ENCARNACIÓN

Research Institute for Symbolic Computation
Johannes Kepler University, A-4040 Linz, Austria
George.Collins@risc.uni-linz.ac.at, mje@acm.org

9 *April* 1995

The paper describes improved techniques for factoring univariate polynomials over the integers. The authors modify the usual linear method for lifting modular polynomial factorizations so that efficient early factor detection can be performed. The new lifting method is universally faster than the classical quadratic method, and is faster than a linear method due to Wang, provided we lift sufficiently high. Early factor detection is made more effective by also testing combinations of modular factors, rather than just single modular factors. Various heuristics are presented that reduce the cost of the factor testing or that increase the chance of successful testing. Both theoretical and empirical computing times are presented.

1. Introduction

Modern algorithms for factoring a univariate polynomial over the integers **Z** are based on some variant of the familiar Berlekamp-Zassenhaus scheme:

(1) For a suitable prime p, compute the factors of the polynomial modulo p;
(2) Lift the factors modulo p to factors modulo p^k for some sufficiently large integer k;
(3) From the factors modulo p^k, determine the irreducible factors over **Z**.

The integer k must be large enough that $p^k \geq B$, where B is a certain bound that will ensure that the irreducible factors over **Z**, also called *true factors*, can be determined from the factors modulo p^k. For most polynomials, the best known bounds B are loose. Motivated by this observation, Wang (1983) introduced the technique of early factor detection, which attempts to determine the true factors while lifting.

Wang's technique can detect only those factors that remain irreducible modulo p. This is a serious limitation since the probability that a random polynomial will remain irreducible modulo p is about $1/n$, where n is the degree of the polynomial (see Knuth (1981), exercise 4.6.2–4). We enhance the effectiveness of early factor detection by introducing an efficient method for also testing combinations of modular factors to determine if they

† This research was supported in part by Austrian FWF project no. P8572-PHY.

have been lifted sufficiently high. We present heuristics that reduce the amount of time spent on unsuccessful factor testing, and that increase the chance that testing will be successful.

Crucial for early factor detection is a lifting method that lifts all the factors *simultaneously*, by which we mean that all factors are lifted to the next level before any factors are lifted further. We describe a new lifting method that lifts factors simultaneously, and which is faster than the classical quadratic lifting method. Wang (1992) also describes a lifting method that lifts factors simultaneously. Our algorithm is faster than Wang's, provided we lift to sufficiently high levels.

In the next section we present our lifting method and compare its theoretical and empirical computing times with those of the classical quadratic method and Wang's method. In Section 3 we describe the various techniques that we use to efficiently test combinations of modular factors. The results of experiments with an implementation are also presented in that section.

2. The New Lifting Method

The lifting method that we present in this section consists of small but important changes to the usual linear Hensel lifting algorithm (see Miola & Yun (1974)) that allow us to perform early factor detection efficiently. We will be comparing our lifting scheme with the classical quadratic method, which does not allow efficient early factor detection, as well as with a linear method due to Wang (1992), which does allow early factor detection.

Let $C \in \mathbf{Z}[x]$ be a primitive and squarefree polynomial to be factored and let p be a prime dividing neither the leading coefficient nor the discriminant of C. Let $C \equiv \prod_{i=1}^{r} A_i \pmod p$ be the complete factorization of C modulo p, where the leading coefficients satisfy $\mathrm{ldcf}(C) \equiv \mathrm{ldcf}(A_1) \pmod p$, and A_i is monic for $i = 2, \ldots, r$. Using the classical quadratic Hensel algorithm, as presented for example by Musser (1971, 1975), we lift this factorization modulo p to a factorization modulo q, where q is the largest power of p that is still single-precision, that is, fits in a single computer word. Let this lifted factorization be $C \equiv \prod_{i=1}^{r} A_{i,1} \pmod q$, and in general let

$$C \equiv \prod_{i=1}^{r} A_{i,j} \pmod{q^j}$$

be the factorization lifted to one modulo q^j. When lifting from modulus p to modulus q, we also compute the *lift basis* polynomials $S_i, T_i \in \mathbf{Z}_q[x]$, $i = 1, \ldots, r-1$, which satisfy

$$A_{i,1}S_i + B_{i,1}T_i \equiv 1 \pmod q,$$

where $B_{i,j} = \prod_{h=i+1}^{r} A_{h,j}$.

A key ingredient of our lifting method is that we compute the q-adic expansion of C; this allows us to lift all the factors to modulus q^{j+1} before lifting any to modulus q^{j+2}, an essential requirement for early factor detection.

For each $m \geq 1$, let $C_m = \phi_{q^m}(C)$ and

$$C_m = \sum_{j=0}^{m-1} D_j q^j, \qquad D_j \in \mathbf{Z}_q[x].$$

We start by lifting the factorization of C modulo q to a factorization modulo q^2 by the usual linear Hensel method, as follows. First compute

$$U_{1,2} := (C_2 - A_{1,1}B_{1,1})/q$$

and then $\tilde{U}_{1,2} := \phi_q(U_{1,2})$. Using the lift basis polynomials S_1 and T_1, solve the congruence $A_{1,1}Y_{1,2} + B_{1,1}Z_{1,2} \equiv \tilde{U}_{1,2} \pmod{q}$ for $Y_{1,2}$, $Z_{1,2} \in \mathbf{Z}_q[x]$, with $\deg(Z_{1,2}) < \deg(A_{1,1})$ and $\deg(Y_{1,2}) < \deg(B_{1,1})$, then set $A_{1,2} := A_{1,1} + qZ_{1,2}$ and $B_{1,2} := B_{1,1} + qY_{1,2}$. Compute $A_{i,2}$ and $B_{i,2}$, for $i = 2, 3, \ldots, r-1$, in like manner, but with $B_{i-1,2}$ in place of C_2. Notice that $C_2 = B_{0,2}$ and $B_{r-1,2} = A_{r,2}$.

Lifting from modulus q^j to modulus q^{j+1}, for $j \geq 2$, is similar but we compute

$$U_{i,j+1} := (B_{i-1,j+1} - A_{i,j}B_{i,j})/q^j$$

differently. Let us first consider the case $i = 1$. Writing $C_{j+1} = C_j + q^j D_j$, $A_{1,j} = A_{1,j-1} + q^{j-1}Z_{1,j}$, and $B_{1,j} = B_{1,j-1} + q^{j-1}Y_{1,j}$, we find after some manipulation that

$$U_{1,j+1} = D_j - V_{1,j} - q^{j-2}Y_{1,j}Z_{1,j},$$

where

$$V_{1,j} := (A_{1,j-1}Y_{1,j} + B_{1,j-1}Z_{1,j} - U_{1,j})/q.$$

(The division will be exact.) Therefore

$$\tilde{U}_{1,3} \equiv D_2 - V_{1,2} - Y_{1,2}Z_{1,2} \pmod{q}$$

and

$$\tilde{U}_{1,j+1} \equiv D_j - V_{1,j} \pmod{q}, \qquad j \geq 3.$$

For $i \geq 2$ the situation is virtually the same, but with $B_{i-1,j+1}$ in place of C_{j+1}. This not really different since $C_{j+1} = B_{0,j+1}$. But whereas $C_{j+1} = C_j + q^j D_j$, we have $B_{i-1,j+1} = B_{i-1,j} + q^j Y_{i-1,j+1}$, so that $Y_{i-1,j+1}$ plays the role of D_j. Indeed, if we define $Y_{0,j+1}$ to be D_j, and define

$$V_{i,j} := (A_{i,j-1}Y_{i,j} + B_{i,j-1}Z_{i,j} - U_{i,j})/q, \tag{2.1}$$

then

$$U_{i,j+1} = Y_{i-1,j+1} - V_{i,j} - q^{j-2}Y_{i,j}Z_{i,j}, \tag{2.2}$$

and therefore

$$\tilde{U}_{i,3} \equiv Y_{i-1,3} - V_{i,2} - Y_{i,2}Z_{i,2} \pmod{q} \tag{2.3}$$

and

$$\tilde{U}_{i,j+1} \equiv Y_{i-1,j+1} - V_{i,j} \pmod{q}, \qquad j \geq 3, \tag{2.4}$$

for $i = 1, \ldots, r-1$.

Having computed $\tilde{U}_{i,j+1}$, we proceed as in the usual linear Hensel algorithm: we solve the congruence

$$A_{i,1}Y_{i,j+1} + B_{i,1}Z_{i,j+1} \equiv \tilde{U}_{i,j+1} \pmod{q} \tag{2.5}$$

for $Y_{i,j+1}$, $Z_{i,j+1} \in \mathbf{Z}_q[x]$, with $\deg(Z_{i,j+1}) < \deg(A_{i,1})$ and $\deg(Y_{i,j+1}) < \deg(B_{i,1})$, and then set $A_{i,j+1} := A_{i,j} + q^j Z_{i,j+1}$ and $B_{i,j+1} := B_{i,j} + q^j Y_{i,j+1}$.

2.1. Theoretical Computing Time

For estimating the running time of this lifting method in number of word operations, let c be the word length of the longest coefficient of C, let $n := \deg(C)$ be the degree of C, let $n_i := \deg(A_i)$, for $i = 1, \ldots, r$, and let $m_i := \deg(B_i) = \sum_{j=i+1}^{r} n_j$, for $i = 1, \ldots, r-1$.

Computing $U_{i,j+1}$ in (2.2) takes $O(jn_i m_i)$ time. Solving the congruence (2.5) takes $O(n_i^2 + m_i^2)$ time. Computing D_j takes $O(jn + cn)$ time. The times for these computations dominate the times for the others so the time for lifting all factors from modulus q^j to q^{j+1} is $O(jn^2 + rn^2 + cn)$. Summing on j, the total time for lifting from modulus q to q^k is

$$O(k^2 n^2 + rkn^2 + kcn). \tag{2.6}$$

Lifting quadratically from modulus p to q, as well as computing the lift basis polynomials, takes $O(rn^2 + cn)$ time. Therefore, the total time for lifting the factorization modulo p to one modulo q^k is also given by (2.6).

In comparison, the classical quadratic method takes

$$O(rk^2 n^2 + kcn)$$

time to lift the factorization modulo p to one modulo q^k. (See Musser (1971) for a detailed analysis.) We see that the time for quadratic lifting strictly dominates the time for the new linear method.

Quadratic lifting methods are not well suited for early factor detection since there are not enough opportunities for testing factors, or combinations thereof. This is especially true of the latter lifting steps, when there is a higher chance of successful factor testing. For this reason, we will be comparing our lifting method with the linear method of Wang (1992), which we will now briefly describe.

Again let $C \equiv \prod_{i=1}^{r} A_{i,j} \pmod{q^j}$. Assume that we have lift basis polynomials $V_i \in \mathbf{Z}_q[x]$, for $i = 1, \ldots, s$, such that

$$V_1 B_1 + V_2 B_2 + \cdots + V_s B_s \equiv 1 \pmod{q},$$

where $B_l = \prod_{i \neq l} A_{i,1}$. Wang's method lifts a factorization modulo q^j to a factorization modulo q^{j+1} by first computing the residue

$$D := \big(C - \prod_{i=1}^{r} A_{i,j} \pmod{q^{j+1}}\big)/q^j. \tag{2.7}$$

The method then computes the correction coefficients $H_i := DV_i \pmod{A_{i,j}}$. Finally the factors are lifted by setting $A_{i,j+1} := A_{i,j} + q^j H_i$.

The time to produce the lift basis for Wang's method is $O(rn^2)$. Computing the residue in (2.7) takes $O(j^2 n^2 + jnc)$ time and this dominates the time for the other computations. Hence the time to lift from modulus p to q^k by Wang's method is

$$O(k^3 n^2 + rn^2 + k^2 nc). \tag{2.8}$$

Comparing (2.6) and (2.8), we can expect our lifting method to be faster than Wang's if we have to lift to a sufficiently high level (k is large). However, we can expect Wang's method to be faster if there are relatively many modular factors (r is large).

f	b	k	r	Q	W	N
2	75	6	5	1.85	0.48	0.55
2	150	12	4	3.72	1.48	1.32
2	300	24	7	7.85	7.15	3.77
2	600	48	6	18.25	39.57	11.10
2	1200	96	3	46.60	160.90	31.68
4	75	6	10	2.32	0.68	0.87
4	150	12	6	3.07	1.72	1.67
4	300	24	9	9.25	9.60	5.13
4	600	48	7	17.45	40.15	12.22
4	1200	96	8	64.57	310.75	48.12
8	75	6	17	2.82	0.95	1.32
8	150	12	12	6.17	2.55	2.55
8	300	24	15	12.48	12.48	7.29
8	600	48	17	33.58	74.50	21.48
8	1200	96	18	88.40	407.54	70.12

Table 1. Comparison of the three lifting methods.

2.2. Empirical Computing Time

The three lifting methods—the classical quadratic method, Wang's method, and the new method—were implemented in SACLIB, a C-language library of algebraic algorithms (Collins *et al.*, 1993). We applied each of the methods to squarefree polynomials of degree 40, which were constructed by multiplying together f randomly generated polynomials of degree $40/f$ with coefficients having bit length at most b, where $f = 2, 4, 8$, and $b = 75, 150, 300, 600, 1200$. These degree-40 polynomials were then factored modulo five primes, none of which divided the leading coefficient or the discriminant of the polynomial. Beginning with the prime 3, successively larger primes were used until five suitable primes were found. The smallest prime yielding the fewest factors was then chosen as the prime to use for lifting.

The results of our experiments are given in Table 1. The modular factors were lifted to factors modulo q^k, where q is the largest single-precision power of the prime used for lifting, and k is approximately the level at which we will be able to recover the coefficients of the true factors (see Section 3), assuming q is about 25 bits long. (Because of the way in which SACLIB handles memory management, the largest single-precision integer in our implementation on a 32-bit machine is $2^{29} - 1$.) The values of k are given in the third column. (Notice that k need not be a power of 2 even when quadratic lifting is used, since for the final quadratic lifting step we can reduce the lift basis polynomials by an appropriate power of q so that the final lifting step lifts to precisely q^k.) The column labeled r gives the number of modular factors being lifted. The columns labeled Q, W, and N give the times for the classical quadratic algorithm, Wang's linear method, and our new method, respectively. These times, measured on a DECstation 5000/240, are in seconds and do not include the time needed for garbage collection.

For these examples, we see that the new method is universally faster than the quadratic method. Compared to Wang's method, the new method is faster when we lift to sufficiently high levels, as predicted by the theoretical analysis presented above.

3. Early Factor Detection Revisited

The technique of early factor detection was introduced by Wang (1979, 1983) with the goal of reducing the number of lifting steps that are performed. During the lifting process, the monic lifted factors are converted, if possible, to congruent monic polynomials with rational number coefficients, which are then used as trial divisors of C, the polynomial being factored. For reconstructing a rational number from its modular residue, an adaption of the extended Euclidean algorithm can be used (Wang, 1981; Wang *et al.*, 1982). The authors have recently described a more efficient algorithm for reconstructing rational numbers (Collins & Encarnación, 1994).

Few of the true factors of C will be found unless products of lifted modular factors are also tested: A random irreducible polynomial of degree n will remain irreducible modulo a prime p with probability only $1/n$, and will split into an average of H_n irreducible modular factors. (The number H_n is the harmonic number $1 + 1/2 + \cdots + 1/n$; see Knuth (1981), exercise 4.6.2–5.) Our algorithm factors C modulo five primes and chooses the smallest prime that produced the fewest modular factors, but this improves the odds only slightly. Therefore, early factor detection can be made more effective by testing not only single modular factors but also products of these factors.

Our algorithm tests single factors, then products of pairs of factors, and so on, up to products of any specified number of modular factors. However, we avoid actually computing these products in almost all cases except those where the product yields a true factor. First of all, we check whether the sum of the degrees of the modular factors being tested is a possible factor degree. We determine the set of possible factor degrees—henceforth referred to as the *factor degree set*—using the method in Musser (1978). If the degree sum is in the factor degree set, then we compute the product of the trailing coefficients of the modular factors being tested. (When forming this product, we make use of partial products from previous computations. For instance, if we are testing the product of the modular factors A_1, A_2, and A_4, and we have already tested the product of A_1, A_2, and A_3, we would already have computed the product of the trailing coefficients of A_1 and A_2. Therefore, forming the product of the trailing coefficients of A_1, A_2, and A_4 will require only one more multiplication, rather than two.) We then attempt to convert this trailing coefficient to a rational number using the algorithm described in Collins & Encarnación (1994). If the conversion is successful, then we check, as Wang (1983) does, if the denominator divides the leading coefficient of C. If the division is exact, then we compute the product of the modular factors being tested and attempt to convert each of the coefficients of the product into rational numbers. If all the coefficients are successfully converted, then we trial divide C by the primitive integral polynomial similar to the rational polynomial.

It should be noticed that the time required for each rational number reconstruction is $O(j^2)$, where q^j is the current modulus. If there are r modular factors, and we test combinations of up to s factors, $s < r$, then the time required is $O(r^s j^2)$. Since the time for lifting at level j using the method described above in Section 2 is only $O(jn^2 + rn^2 + cn)$, there is the danger that if s is too large, then the time spent on early factor detection will be larger than the lifting time saved. But if s is too small, then some factors will not be recovered as soon as would otherwise be possible, and lifting time will be wasted.

Another concern is to begin testing at a level at which we have a reasonable expectation that testing will be successful. If we start too early, then we waste time on unsuccessful testing. If we start too late, then we defeat the purpose of early factor detection, which is

	f	10	20	30
	1	13	8	3
$d = 10$	2	76	89	95
	3	11	3	2
	time	*2.2*	*5.5*	*10.1*

	f	10	20	30
	1	17	10	2
$d = 25$	2	73	86	96
	3	10	4	2
	time	*5.5*	*13.8*	*25.2*

Table 2. Estimates for two-factor polynomials.

	f	10	20	30
	2	20	15	5
$d = 10$	3	62	71	88
	4	17	14	7
	5	1	0	0
	time	*4.4*	*10.4*	*19.0*

	f	10	20	30
	2	22	17	6
$d = 25$	3	55	70	87
	4	23	13	7
	time	*11.5*	*26.2*	*47.3*

Table 3. Estimates for three-factor polynomials.

to avoid unnecessary lifting. We will presently investigate three heuristics to guide us in choosing how many (and which) factors to combine, and at which lifting level to perform factor testing; before doing so, we will first discuss various topics that relate to early factor detection.

3.1. Estimating the Number of True Factors

The heuristics discussed below rely on an estimate of the number of true factors; computing such an estimate is the topic of this section.

The average number of linear factors of C modulo a random prime p is equal to the number of true factors of C, as $p \to \infty$ (see Knuth (1981), exercise 4.6.2–38). Although this is an asymptotic statement, we will proceed as if it were true also for small primes; the experimental results described below suggest that this is not unreasonable. We thus take f, our estimate of the number of true factors, to be the average number of linear factors of C modulo several small primes. We compute the number of linear factors of C modulo a prime p by simply counting the number of roots of C in the field of integers modulo p, which we can do by evaluation at a cost of $O(nw+np)$, where $n = \deg(C)$ and w is the word length of the max norm of C. The number of true factors can be determined exactly if we compute the average number of linear factors modulo sufficiently many primes (Weinberger, 1984). Unfortunately, "sufficiently many" is much too large to be practical. The question then is: how many primes do we use?

To help answer this question, we generated six sets of 100 polynomials having two, three, or four true factors, each having degree $d = 10$ or $d = 25$, respectively, and coefficients at most 100 bits long. We computed the number of linear factors of each polynomial modulo 10, 20, and 30 successively larger primes beginning with the prime 3. Primes dividing the leading coefficient of the polynomial were skipped. To avoid problems arising from the presence of multiple roots, we discarded and replaced those primes for

	f	10	20	30
$d = 10$	2	16	0	0
	3	53	12	13
	4	53	71	74
	5	24	17	13
	6	5	0	0
	time	*8.0*	*17.2*	*30.4*

	f	10	20	30
$d = 25$	2	1	0	0
	3	20	19	13
	4	55	64	74
	5	18	17	13
	6	6	0	0
	time	*19.7*	*43.8*	*75.4*

Table 4. Estimates for four-factor polynomials.

which the polynomials had multiple roots. The results of our experiment are displayed in Tables 2, 3, and 4, the entries of which are the number of times, out of 100, that we estimated that we had f true factors, where the values of f are given in the first column. These values of f were obtained as the average number of linear factors rounded to the nearest integer, with round-to-even to break ties. The headings of the second, third, and fourth columns give the number of primes that were used for the estimate. The italicized entries in the last row are the times, in seconds, required by the computations. Based on these tables, we decided to use 25 primes for estimating the number of true factors.

3.2. Computing a Lifting Bound

Let $C = C_1C_2\cdots C_t$ be a factorization of C, where $t \geq 2$. Beauzamy *et al.* (1993) show that at least one of the factors, say C_1, satisfies

$$|C_1|_\infty \leq \frac{2^{5/8}}{\pi^{3/8}}\, e^{1/4n}\, \frac{2^{n/2}}{n^{3/8}}[C]_2^{1/t}, \tag{3.1}$$

where $[C]_2$ is the weighted norm

$$[C]_2 = \left(\sum_{i=0}^{n} \frac{|c_i|^2}{\binom{n}{i}}\right)^{1/2},$$

in which c_i is the ith coefficient of C. The algorithm for reconstructing rationals requires that the modulus be larger than twice the square of $|C_1|_\infty$, so we are interested in twice the square of the right-hand side of inequality (3.1), which we overestimate with

$$\left\lceil (6/5)2^{n+1}\left\lceil\Big(\sum_{i=0}^{n}\Big\lceil\frac{|c_i|^2}{\binom{n}{i}}\Big\rceil\Big)^{1/t}\right\rceil \Big/ \lfloor n^{3/4}\rfloor\right\rceil, \tag{3.2}$$

avoiding rational arithmetic in the sum. In deriving this overestimate we assumed that $n \geq 3$ since quadratic polynomials can be factored using the quadratic formula. We will refer to (3.2), evaluated at $t = 2$, as the *two-factor lifting bound* for the polynomial C.

Beauzamy *et al.* (1993) suggest using a method based on Pascal's triangle to compute the binomial coefficients $\binom{n}{i}$. However, that method requires $O(n^2)$ additions. Instead, we use a method that is based on the recurrence

$$\binom{n}{i+1} = \frac{n-i}{i+1}\binom{n}{i}, \qquad i = 0, \ldots, n-1,$$

which requires only n multiplications and n divisions.

3.3. Ensuring Irreducibility

One problem that we have when testing products of two or more modular factors that we do not have when testing only single modular factors is that of ensuring the irreducibility (over the integers) of the product. It may happen, albeit rarely, that some reducible proper factor A of the polynomial we are factoring has an irreducible factor B with some coefficients that are larger than any coefficient of A. In this case, the factor A may be recoverable at a lower lifting level than B. To solve this problem, our algorithm computes the two-factor lifting bound for the product before doing the trial division (*cf.* Beauzamy *et al.* (1993)). If the current modulus is larger than this bound, then we know that the product must be irreducible over the integers, should it be a divisor of the polynomial we are factoring. For quadratic factors, the algorithm uses the quadratic formula to ensure irreducibility, or to factor the polynomial when appropriate.

3.4. Recomputing the Factor Degree Set

When a true factor is found, we recompute the factor degree set for the remaining polynomial to be factored as follows. Let $G_1G_2\cdots G_t \equiv C \pmod p$ be the distinct-degree factorization of C modulo p, where G_i is the product of all the irreducible factors of C modulo p of degree i. Let $C^* := C/\tilde{C}$, where $\tilde{C}$ is the true factor we found. Then the distinct-degree factorization of C^* modulo p will be $H_1H_2\cdots H_t \equiv C^* \pmod p$, where $H_i := G_i/\gcd(G_i, \tilde{C})$. We thus compute the distinct-degree factorization of the remaining polynomial modulo each of the five primes used for the modular factorizations in the manner just described, and determine the factor degree set of the remaining polynomial using the method in Musser (1978).

3.5. Termination

The algorithm will terminate when it determines that the remaining factor (after dividing out any true factors that are found) is irreducible. This will happen in one of three ways: (a) The current modulus is larger than the two-factor lifting bound of the remaining polynomial, and the algorithm has tested all combinations of up to $r-1$ modular factors, where r is the number of remaining modular factors. (We may need to combine as many as $r-1$ modular factors since it could happen that the factor with the smallest coefficients factored into $r-1$ modular factors.) (b) The factor degree set contains only 0 and the degree of the remaining polynomial. (c) The remaining polynomial is quadratic, in which case the quadratic formula is applied. Our experience with numerous examples indicates that termination is effected by method (b) more often than not.

3.6. Heuristics for Factor Testing

In the descriptions of the heuristics we will be comparing we often refer to *unsuccessful testing time*, which we define as follows. When we perform factor testing at any particular level, we will either find true factors or we will not. If no true factors are found, then all the time spent on factor testing will be added to the unsuccessful testing time. This time includes the time for forming products of modular trailing coefficients, reconstructing rationals, dividing the leading coefficient by denominators, and possibly doing trial divisions that fail, though such trial divisions are rare. If at least one true factor is found at a certain level, then none of the time spent on factor testing at that level is added to unsuccessful testing time.

We assume that we have computed an estimate f of the number of true factors as discussed above, with the modification that if $f < 2$, then we set $f := 2$. Given this estimate, we can get an estimate s of the average number of modular factors of each true factor by setting $s := r/f$, rounded to the nearest integer, with round-to-even to break ties, where r is the total number of modular factors.

We will compare empirically the following three heuristics:

Heuristic 1. Combine as many as s modular factors. Perform factor testing on the factors modulo q, and start factor testing again as soon as the modulus is larger than twice the square of the fth root of the max norm of C.

Heuristic 2. Combine as many as $s+2$ modular factors. Test after each lifting step as long as the time spent on unsuccessful factor testing is no larger than a fourth of the total time spent on linear lifting. If the unsuccessful testing time becomes larger than a fourth of the lifting time, then refrain from testing until the testing time again drops to at most a fourth of the lifting time. However, regardless of the time spent on unsuccessful testing, test at the first level for which the modulus is larger than twice the square of the fth root of the max norm of C.

Heuristic 3. Let c be the max norm of C and let k be the smallest integer such that $q^k \geq 2c^2$. At level j, that is, when the current modulus is q^j, test all products of modular factors that are such that the degree d of the product satisfies $d \leq (j/k)n$, where $n = \deg(C)$. As in the previous heuristic, we refrain from testing whenever unsuccessful testing time becomes larger than a fourth of total linear lifting time, but we test at the first level for which the modulus is larger than twice the square of the fth root of the max norm of C, regardless of unsuccessful testing time.

Each heuristic will be in force only until we have lifted to the two-factor lifting bound of C, at which point we test all possible combinations of modular factors.

Before presenting the results of our comparison, we will first give the rationale behind these heuristics. In the first heuristic, we suppose that the max norms of the true factors do not differ by much. The factors modulo q are tested to detect early in the lifting process small factors, such as $x-1$, which are not uncommon in applications. In the second heuristic, we combine two more modular factors than the estimated average since we can expect that some true factors will have more than the average number of modular factors. To prevent too much time being spent on unsuccessful factor testing, we place a cap on this time relative to the total lifting time; the ratio 4 was somewhat arbitrarily chosen. We test at the first level for which the modulus is larger than twice the square of the fth root of the max norm of C since the stipulation that unsuccessful testing time be no more than a fourth of lifting time may otherwise force us to lift well beyond this level, a level at which we are likely to find true factors. In the third heuristic, we suppose that the coefficient sizes of the true factors are roughly proportional to their degrees. For example, polynomials arising in the projection phase of cylindrical algebraic decomposition (Collins, 1975) behave in this way.

Each of the three heuristics was implemented in SACLIB and applied to randomly generated polynomials that were products of either two or three true factors. Table 5 gives the unsuccessful testing time and the lifting time for each of the three heuristics applied to polynomials with two true factors, which had degrees d_1 and d_2 and coefficients at most $10c_1$ and $10c_2$ bits long. The times, given in milliseconds, are averages for ten

		heuristic 1			heuristic 2			heuristic 3		
(d_1, d_2)	(c_1, c_2)	testing	lifting		testing	lifting		testing	lifting	
(10, 10)	(10, 10)	2	195	*8.0*	32	198	*8.0*	17	192	*8.0*
(20, 20)	(20, 20)	12	1648	*15.3*	321	1660	*15.3*	137	1655	*15.3*
(30, 30)	(30, 30)	10	6158	*23.5*	1217	6159	*23.5*	776	6087	*23.5*
(10, 20)	(10, 10)	2	342	*8.1*	64	336	*8.0*	32	330	*8.0*
(15, 30)	(15, 15)	17	1222	*11.7*	164	1248	*11.8*	74	1234	*11.7*
(20, 40)	(20, 20)	7	3108	*15.3*	493	3074	*15.3*	195	3081	*15.3*
(10, 20)	(10, 20)	3	582	*11.3*	40	355	*8.2*	10	362	*8.2*
(15, 30)	(15, 30)	28	2270	*17.5*	222	1277	*11.7*	52	1270	*11.7*
(20, 40)	(20, 40)	63	5861	*23.2*	668	3454	*16.2*	211	3183	*15.4*

Table 5. Comparison of the heuristics for two-factor polynomials.

		heuristic 1			heuristic 2			heuristic 3		
(d_1, d_2, d_3)	(c_1, c_2, c_3)	testing	lifting		testing	lifting		testing	lifting	
(10, 10, 10)	(10, 10, 10)	5	535	*8.8*	126	467	*8.2*	62	486	*8.1*
(15, 15, 15)	(15, 15, 15)	3	1525	*11.7*	432	1522	*11.7*	117	1543	*11.7*
(20, 20, 20)	(20, 20, 20)	13	4291	*16.4*	1042	3944	*15.6*	473	3950	*15.6*
(5, 10, 15)	(10, 10, 10)	5	434	*8.3*	121	444	*8.3*	40	445	*8.3*
(10, 20, 30)	(20, 20, 20)	40	4005	*16.2*	1072	3932	*15.9*	640	3725	*15.4*
(15, 30, 45)	(30, 30, 30)	58	14522	*23.3*	5866	14477	*23.3*	3310	14487	*23.3*
(5, 10, 15)	(5, 10, 15)	7	512	*9.0*	72	423	*8.0*	10	427	*8.0*
(10, 20, 30)	(10, 20, 30)	37	4920	*19.2*	642	3652	*15.7*	148	3626	*15.6*
(15, 30, 45)	(15, 30, 45)	657	18118	*27.2*	2647	13479	*22.8*	793	13490	*22.8*

Table 6. Comparison of the heuristics for three-factor polynomials.

polynomials. The italicized entries are the average maximum lifting levels. Table 6 gives similar data for polynomials with three true factors.

The average lifting times and levels in rows 1 through 6 of Table 5 do not suggest any significant differences between the heuristics, but we see that heuristics 2 and 3 spend more time on testing than heuristic 1, with heuristic 3 spending less time on testing than heuristic 2. For rows 7 through 9, however, we see that heuristics 2 and 3 are significantly better than heuristic 1, and, again, heuristic 3 is spending less time on unsuccessful testing than heuristic 2. Similar remarks apply to Table 6.

Presenting only averages, Tables 5 and 6 do not reveal the variation from polynomial to polynomial. To illustrate this variation we display in Tables 7, 8, and 9, respectively, the performance of the three heuristics on each of the ten polynomials corresponding to row 6 of Table 6. Each row in the tables gives data for one polynomial. For all ten polynomials, we correctly estimated that there were 3 true factors. Column r gives the number of factors that the polynomial had modulo the prime used for lifting. Column s (column $s + 2$) in Table 7 (Table 8) gives the maximum number of modular factors that were combined. The columns labeled 'mod. prod. count' give the total number of products of trailing coefficients that were computed, and those labeled 'mod. prod. time' give the corresponding computing time. The columns 'rat. recon. count' and 'rat. recon. time', respectively, give the total number of rational reconstructions of products of modular

#	r	s	mod. prod. count	time	rat. recon. count	time	testing	lifting	level	total
1	12	4	1075	66	783	50	267	14379	22	23832
2	9	3	172	0	129	0	50	13832	22	21667
3	9	3	143	0	105	0	17	11851	22	19266
4	10	3	228	0	175	0	50	17733	25	25117
5	8	3	111	0	79	0	33	12383	22	19532
6	9	3	172	17	129	0	67	16267	24	24183
7	7	2	34	17	28	0	17	14202	24	22565
8	9	3	172	0	129	0	50	14331	23	22315
9	12	4	1088	51	793	17	267	17300	24	26949
10	8	3	126	17	92	0	34	13835	25	22068

Table 7. Detail of heuristic 1 for $(d_1, d_2, d_3) = (15, 30, 45)$, $(c_1, c_2, c_3) = (30, 30, 30)$.

#	r	$s+2$	mod. prod. count	time	rat. recon. count	time	testing	lifting	level	total
1	12	6	8152	1847	4998	1501	4765	14665	22	28598
2	9	5	3792	1673	2286	2014	4285	13950	22	25817
3	9	5	3432	1682	2010	1238	3833	11616	22	22868
4	10	5	5090	2888	3185	2496	6566	17490	25	31436
5	8	5	3177	1326	1773	1388	3383	12384	22	23000
6	9	5	3792	1949	2286	1754	4498	16270	24	28484
7	7	4	2844	1162	1764	1449	3365	14218	24	25751
8	9	5	3792	1546	2286	1739	4200	14150	23	26616
9	12	6	8178	1787	5018	1833	5367	17098	24	31915
10	8	5	3008	1479	1744	1626	3885	13848	25	26051

Table 8. Detail of heuristic 2 for $(d_1, d_2, d_3) = (15, 30, 45)$, $(c_1, c_2, c_3) = (30, 30, 30)$.

trailing coefficients that were performed and the total computing time required for these reconstructions.

We chose heuristic 3 as the best strategy for our purposes, but a word of caution is in order: If one expects that the true factors of the polynomial one is factoring deviate significantly from the presupposition that coefficient sizes are proportional to degrees, then one is probably better off using a different heuristic.

In Tables 10 and 11, respectively, we give the results of our experiments comparing single- and multiple-factor testing applied to two-factor and three-factor polynomials. Single-factor testing is performed after each lifting step until we reach the single-factor bound of the polynomial, when we test all possible combinations of modular factors. Multiple-factor testing is performed using heuristic 3.

In the first six rows of Table 10 we do not see a significant difference between single- and multiple-factor testing. This is to be expected since an exhaustive search for true factors will be performed by either algorithm as soon as the modulus becomes larger than the two-factor lifting bound of the polynomial, which, for these polynomials, is not much larger than the square of the max norms of the two true factors. Looking at the other rows in Table 10 and those in Table 11, we see that multiple-factor testing can be noticeably better than single-factor testing.

#	r	mod. prod. count	mod. prod. time	rat. recon. count	rat. recon. time	testing	lifting	level	total
1	12	4264	2769	2291	2500	6431	14500	22	29265
2	9	2097	1676	1150	1777	3800	13751	22	25099
3	9	2241	1662	1161	1239	3268	11801	22	22584
4	10	2655	1879	1471	2090	4533	17269	25	29169
5	8	1163	914	600	801	2082	12202	22	21533
6	9	1870	1599	1124	1720	3850	15962	24	27266
7	7	617	333	370	500	1003	14279	24	23182
8	9	2323	1681	1266	1604	3834	13947	23	25934
9	12	3692	2018	2024	2029	4965	16984	24	31150
10	8	1659	1314	927	1152	2849	13634	25	24583

Table 9. Detail of heuristic 3 for $(d_1, d_2, d_3) = (15, 30, 45)$, $(c_1, c_2, c_3) = (30, 30, 30)$.

(d_1, d_2)	(c_1, c_2)	single-factor testing: testing	lifting		total	multiple-factor testing: testing	lifting		total
(10, 10)	(10, 10)	12	177	*8.0*	432	17	188	*8.0*	508
(20, 20)	(20, 20)	44	1780	*16.3*	2775	158	1612	*15.3*	2888
(30, 30)	(30, 30)	164	6459	*24.5*	9016	746	6010	*23.5*	9474
(10, 20)	(10, 10)	14	355	*8.4*	785	30	333	*8.0*	877
(15, 30)	(15, 15)	35	1416	*12.9*	2605	90	1224	*11.8*	2620
(20, 40)	(20, 20)	81	3470	*16.8*	5745	208	3054	*15.3*	5600
(10, 20)	(10, 20)	20	524	*10.4*	1012	15	365	*8.2*	952
(15, 30)	(15, 30)	73	2057	*16.7*	3307	54	1246	*11.7*	2627
(20, 40)	(20, 40)	163	5668	*22.7*	8225	205	3146	*15.4*	5918

Table 10. Single- vs. multiple-factor testing for two-factor polynomials.

(d_1, d_2, d_3)	(c_1, c_2, c_3)	single-factor testing: testing	lifting		total	multiple-factor testing: testing	lifting		total
(10, 10, 10)	(10, 10, 10)	40	637	*10.0*	1278	65	460	*8.1*	1255
(15, 15, 15)	(15, 15, 15)	100	2340	*15.7*	3745	135	1510	*11.7*	3152
(20, 20, 20)	(20, 20, 20)	245	7158	*23.3*	10175	473	3874	*15.6*	7388
(5, 10, 15)	(10, 10, 10)	32	540	*9.5*	1153	43	442	*8.3*	1207
(10, 20, 30)	(20, 20, 20)	226	6290	*21.8*	9281	649	3670	*15.4*	7250
(15, 30, 45)	(30, 30, 30)	627	25199	*33.0*	33767	3483	14382	*23.3*	25700
(5, 10, 15)	(5, 10, 15)	34	572	*9.9*	1174	16	424	*8.0*	1123
(10, 20, 30)	(10, 20, 30)	221	7069	*24.2*	10060	185	3671	*15.6*	6830
(15, 30, 45)	(15, 30, 45)	731	27357	*35.8*	36146	837	13583	*22.8*	22081

Table 11. Single- vs. multiple-factor testing for three-factor polynomials.

The authors would like to thank Wieb Bosma, John Cannon, and Allan Steel for carefully reading a preliminary draft of this paper.

References

B. Beauzamy, V. Trevisan, and P. S. Wang. (1993). Polynomial factorization: Sharp bounds, efficient algorithms. *J. Symbolic Computation* **15**, 393–413.

G. E. Collins. (1975). Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages, 2nd GI Conference*, *Lecture Notes in Computer Science* **33**, 134–183, Berlin, Springer-Verlag.

Collins, G. E. *et al.* (1993). SACLIB 1.1 User's Guide. Technical Report 93-19, RISC-Linz, Johannes Kepler University, A-4040 Linz, Austria.

G. E. Collins and M. J. Encarnación. (1994). Efficient rational number reconstruction. Technical Report 94-64, RISC-Linz, Johannes Kepler University, A-4040 Linz, Austria.

D. E. Knuth. (1981). *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms.* Addison-Wesley.

A. Miola and D. Y. Y. Yun. (1974). Computational aspects of Hensel-type univariate polynomial greatest common divisor algorithms. In *Proc. EUROSAM '74*, *SIGSAM Bulletin* **8**, 46–54.

D. R. Musser. (1971). *Algorithms for Polynomial Factorization.* PhD thesis, University of Wisconsin, Madison.

D. R. Musser. (1975). Multivariate polynomial factorization. *J. ACM* **22**(2), 291–308.

D. R. Musser. (1978). On the efficiency of a polynomial irreducibility test. *J. ACM* **25**(2), 271–282.

P. S. Wang. (1979). Parallel p-adic construction in the univariate polynomial factoring algorithm. In *Proc. MACSYMA Users' Conference*, 310–318, Cambridge, MA, MIT.

P. S. Wang. (1981). A p-adic algorithm for univariate partial fractions. In *Proc. SYMSAC '81*, 212–217. ACM Press.

P. S. Wang. (1983). Early detection of true factors in univariate polynomial factorization. In *Proc. EUROCAL '83*, *Lecture Notes in Computer Science* **162**, 225–235. Springer-Verlag.

P. S. Wang. (1992). Parallel univariate p-adic lifting on shared-memory multiprocessors. Tech. Rep. ICM-9201-25, Institute for Computational Mathematics, Kent State University, Kent, OH 44242.

P. S. Wang, M. J. T. Guy, and J. H. Davenport. (1982). p-adic reconstruction of rational numbers. *SIGSAM Bulletin* **16**, 2–3.

P. J. Weinberger. (1984). Finding the number of factors of a polynomial. *J. Algorithms* **5**, 180–186.