Computing with Polynomials Given By Black Boxes for Their Evaluations: Greatest Common Divisors, Factorization, Separation of Numerators and Denominators*

ERICH KALTOFEN

Rensselaer Polytechnic Institute, Department of Computer Science, Troy, New York 12180-3590; kaltofen@cs.rpi.edu

BARRY M. TRAGER

I.B.M., T. J. Watson Research Center, Yorktown Heights, New York 10598; bmt@ibm.com

Algorithms are developed that adopt a novel implicit representation for multivariate polynomials and rational functions with rational coefficients, that of black boxes for their evaluation. We show that within this representation the polynomial greatest common divisor and factorization problems, as well as the problem of extracting the numerator and denominator of a rational function, can all be solved in random polynomial-time. Since we can convert black boxes efficiently to sparse format, problems with sparse solutions, e.g., sparse polynomial factorization and sparse multivariate rational function interpolation, are also in random polynomial time. Moreover, the black box representation is one of the most space efficient implicit representations that we know. Therefore, the output programs can be easily distributed over a network of processors for further manipulation, such as sparse interpolation.

1. Introduction

We introduce algorithms that manipulate multivariate polynomials and rational functions that are given by "black boxes": a black box is an object which takes as input a value

*This material is based on work supported by the National Science Foundation under Grants No. CCR-87-05363 and No. CDA-88-05910, and by an IBM faculty development award (first author). A preliminary version of this paper appears in the *Proceedings of the 29th Symposium on Foundations of Computer Science*, IEEE, pp. 296–305 (1988).

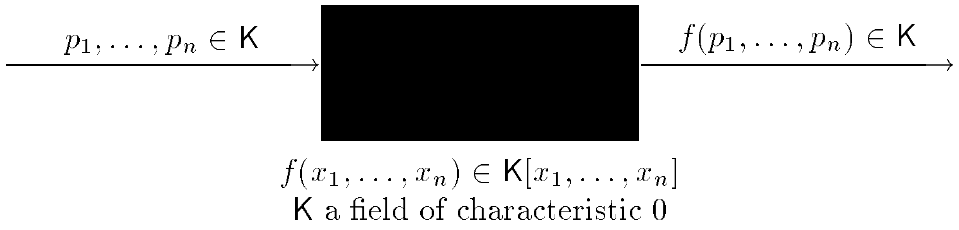


Figure 1: Black box representation of a polynomial.

for each variable, and then produces the value of the polynomial or rational function it represents at the specified point (see Figure 1).

The algorithms' outputs are procedures which will evaluate the answer polynomials at arbitrary points (supplied as the input). These procedures make oracle calls to the black boxes given as the input to the algorithm to evaluate them at certain points dependent on the inputs to the procedure. One result is that we can construct an evaluation procedure for the greatest common divisor of a set of polynomials given their black box representations. A main result is the construction of an evaluation program for all irreducible factors of a black box polynomial (see Figure 2).

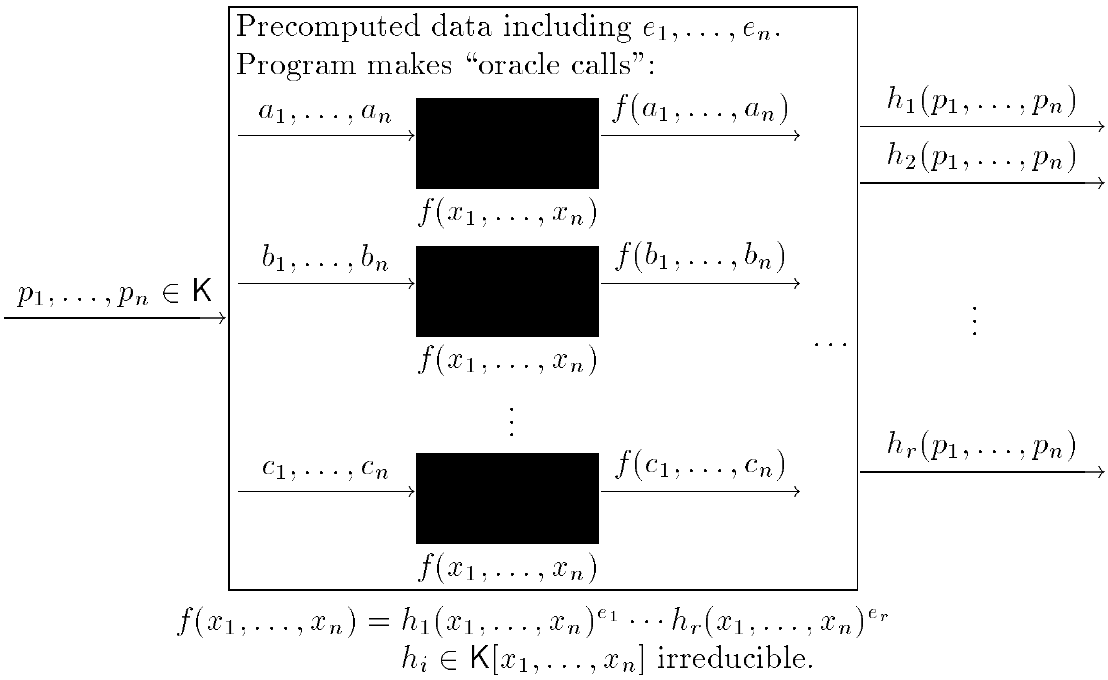


Figure 2: The program for evaluating the irreducible factors of a black box polynomial.

Finally, from a black box for a multivariate rational function we construct an evaluation box for both its reduced numerator and denominator. All three algorithms normalize the constructed polynomials in a deterministic fashion, that is querying the resulting boxes repeatedly will evaluate one and the same associate (scalar multiple) of the goal polynomial. Moreover, for rational coefficients the constructions of the new evaluation boxes are accomplished in polynomial-time with respect to total degree and coefficient size of the given polynomials. The algorithms presented are Monte-Carlo: that is their output—the constructed evaluation programs—procedures are correct with controllably high probability.

Previous results on manipulating implicitly represented multivariate polynomials and rational functions consider straight-line programs as the representation model (Baur and Strassen 1983), (Valiant 1982), (von zur Gathen 1985), (Kaltofen 1986b, 1987, and 1989). The polynomial factorization result for straight-line programs, for instance, performs transformations on each individual assignment in the input program, and cannot be generalized to the black box representation in a straightforward fashion. We employ a new idea for this problem by introducing an extra variable for two different purposes. When setting that variable to 0, the factors map to precomputed fixed univariate factors. We thus can not only identify the bivariate factors independently of the evaluation that is requested, but also normalize them with respect to scalar multiplication. The other purpose of this second variable is that by setting it to 1 we obtain the values of the factors at a given point.

It is, of course, crucial that the program for the irreducible factors evaluates a fixed associate for each multivariate factor. Moreover, the program is with controllably high probability correct, that is it then will always return the correct evaluations of the factors, independently of the input values an adversary may have chosen for the variables. We also impose this requirement on the construction of a program for the numerator and denominator of a rational function.

The solution for numerator and denominator separation found for the straight-line model (Kaltofen 1988) requires stepping through each instruction. Our innovation first uses a mixed radix Padé approximation (see, for instance (von zur Gathen 1986)), that replaces the Taylor series approximation by interpolation. Second, it again makes use of the idea of a multipurpose second variable. However, for that problem the construction of the evaluation procedure for numerators and denominators is substantially more complex. The reason is that one has to deal with zero points of the denominator. Our assumption is that the black box for the rational function indicates in its output if one tries to evaluate at a root of the denominator, and produces a value at all other points. We remark that if the program for evaluating the numerator and denominator is constructed correctly by our algorithm, that with high probability, it will find the values of both the numerator and the denominator polynomials at all values for the variables, even those that are zeros for the denominator.

Our constructed programs with oracle calls to the input black boxes are in many ways superior to the straight-line program model. From a theoretical point of view, these programs can be rapidly converted to sparse format using any of the new sparse polynomial interpolation algorithms (Ben-Or and Tiwari 1988), (Zippel 1990), (Kaltofen and Lakshman 1988), and (Grigoryev et al. 1988). Our black box factorizer thus has, for example, resurrected interpolation as a means of factoring polynomials. That method was last mentioned in the first edition of Knuth's book (1981), but gave way to Hensel lifting for reason of exponential combinatorial blow-up in the identification problem of the factor images.

Having overcome this problem here, we can use sparse interpolation to an added advantage, namely we can distribute the interpolation process over a series of processors using Ben-Or's and Tiwari's (1988) algorithm. Since the polynomials are given by black boxes, the amount of data that needs to be sent to the individual processors is very small, which makes such a distributed computation scheme quite practical. From a theoretical point of view, we also obtain a randomized $\mathcal{NC}$ reduction (Cook 1985) for computing the sparse factors of multivariate polynomials to the problem of factoring univariate polynomials over the coefficient field, resolving an open problem in (von zur Gathen and Kaltofen 1985). Over the rational numbers we get, for example, a randomized $\mathcal{NC}$ algorithm for computing all sparse factors of a multivariate polynomial that have fixed degree.

We can also apply sparse interpolation to the program that we obtain for the numerator and denominator of a black box rational function. Furthermore, if we are given correct upper bounds for the number of non-zero monomials in the numerator and denominator, then by a deterministic zero test (Zippel 1990) we can verify the computed sparse numerator and denominator. Therefore we have a Las Vegas randomized polynomial-time solution to the sparse rational interpolation problem, that is one that always returns the correct answer and has expected polynomial-time complexity.

Before discussing our results, let us mention a linear algebra setting in which black box representation also can be successfully applied. We are given the coefficient matrix A of a linear system in terms of a black box that will multiply this matrix with a chosen vector y (see Figure 3). For simplicity, assume that $A \in \mathsf{K}^{n\times n}$ is non-singular. Wiedemann (1986) presents a Las Vegas randomized algorithm that will solve $Ax = b$ for x, where $x, b \in \mathsf{K}^{n\times 1}$, that requires an expected number of $O(n)$ applications of the black box for A and $O(n^2)$ additional arithmetic operations in K. For the singular case, Wiedemann shows how to compute a random vector in the solution manifold in the same expected number of steps. Furthermore, Wiedemann gives a Las Vegas randomized algorithm that computes the determinant of A in $O(n)$ expected number applications of the black box for A and an additional $O(n^2 \log(n))$ arithmetic steps in K.

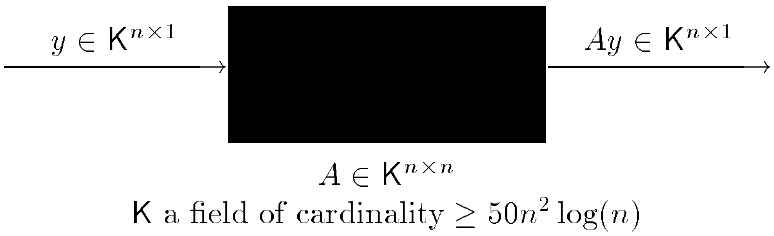


Figure 3: Black box representation of a matrix.

It is again important to realize that Wiedemann's results generalize sparse matrix representation. Clearly, for sparse matrices, the product Ay may only take $O(n)$ arithmetic steps in K. In that case the methods improve on any $O(n^3)$ algorithms for system solving. However, the algorithms also apply to dense matrices with special structure, such as Toeplitz, Vandermonde, or resultant matrices. In (Canny et al. 1989), e.g., we present an efficient algorithm to multiply a Macaulay (1916) resultant matrix and a vector, and thus obtain by Wiedemann's algorithms the most efficient solution known for computing the Macaulay resultant itself. The black box representation of a matrix that allows its multiplication with a given vector is of course a ubiquitous concept in numerical sparse linear system solving, such as in the Lanczos and conjugate gradient methods for solving positive definite sparse linear systems (see (Golub and van Loan 1987)). Wiedemann's algorithms are distinguished from these methods in that they are algebraic and compute an exact answer in contrast to a numerical approximation of the solution.

Notation: We use the symbols := and =: to define new mathematical objects (the new quantities being on the side of the colon), and we use the symbols $\leftarrow$ and $\rightarrow$ as assignment operators in program code. Furthermore, $\deg(f)$ always denotes the total degree of the multivariate polynomial f.

2. Factorization of Polynomials in Black Box Representation

We now present the algorithm that takes a black box for a multivariate polynomial and produces a program that will evaluate all individual irreducible factors at arbitrary values for the variables.

Algorithm *Black Box Polynomial Factorization*

Input: A black box that evaluates $f(x_1, \ldots, x_n) \in \mathsf{K}[x_1, \ldots, x_n]$, where K is a field of characteristic zero. We also assume that we have an effective polynomial factorization algorithm for $\mathsf{K}[y]$ (see Figure 1). Furthermore, a failure probability $\epsilon \ll 1$ is input.

Output: Assume that

$$f(x_1, \ldots, x_n) = \prod_{i=1}^{r} h_i(x_1, \ldots, x_n)^{e_i}, \quad e_i \geq 1,$$

is the factorization of f into irreducibles. First we output integers $\bar{e}_1, \ldots, \bar{e}_n$ such that with probability no less than $1 - \epsilon$, $e_i = \bar{e}_i$ for all $1 \leq i \leq r$.

We also output a program (see Figure 2) that makes oracle calls to the black box for f and has with probability at least $1 - \epsilon$ the following property. The program accepts as input n arbitrary elements $p_1, \ldots, p_n$ in K. It returns the values

$$h_1(p_1, \ldots, p_n), h_2(p_1, \ldots, p_n), \ldots, h_r(p_1, \ldots, p_n) \in \mathsf{K}.$$

Notice that the h_i are determined only up to a multiple in K. The constructed program once and for all chooses an associate for each factor h_i and, for repeated invocations with different arguments, returns the value of that associate. Notice also that the failure probability applies to the construction and not to the execution of the program. That is, with probability at least $1 - \epsilon$ the output program is correct; a correct program *always* produces the true values of the factors.

Step 1: *Pick random field elements*

$$a_2, \ldots, a_n, b_1, \ldots, b_n, c_1, c_3, \ldots, c_n$$

from a sufficiently large finite subset $R \subset \mathsf{K}$. We will give the cardinality of this set in relation to $\deg(f)$ and ϵ in the statement of theorem 1 below.

By standard interpolation compute

$$f_2(X_1, X_2) := f(X_1 + c_1X_2 + b_1, a_2X_1 + X_2 + b_2, \\ a_3X_1 + c_3X_2 + b_3, \ldots, a_nX_1 + c_nX_2 + b_n).$$

The interpolation algorithm needs to know $\deg(f)$. Either $\deg(f)$ or an upper bound is also supplied as input, or it can be probabilistically guessed beforehand as follows (see also (Kaltofen 1988)). First, we will interpolate $f_1(X_1) := f_2(X_1, 0)$. With high probability, depending on the cardinality of R, we have $\deg(f) = \deg(f_1)$. To find $\deg(f_1)$ we compute a succession of polynomials $f_1^{(d)}$ for $d = 1, 2, 3, \ldots$ until $f_1^{(d)} = f_1$, where $f_1^{(d)}(X_1)$ is the interpolate at $X_1 = 0, 1, \ldots, d$ of $f_1(X_1)$. We test whether $f_1^{(d)} = f_1$ by evaluation at a random $A \in R$: if $f_1^{(d)}(A) = f_1(A)$ then we declare $f_1 = f_1^{(d)}$, and $\deg(f_1) = d$. As discussed in (Freeman et al 1986, procedure **StraightDegree**), the $f_1^{(d)}$ can be computed incrementally by using a divided difference table.*

The probability that this method determines $\deg(f)$ correctly depends on the relation of $\deg(f)$ to the cardinality of the set R, from which the random elements are taken. Since the algorithm chooses the set R, say integers of no more than l bits, where l is appropriately selected, some upper bound for $\deg(f)$ must be known to the algorithm beforehand. This is unavoidable since the black-box could compute a polynomial in which all l-bit integer translation points are 'unlucky'. For example, for all such integers, the projected polynomial f_2 might be the zero polynomial, hence unusable. In any case, the failure probability is proportional to $\deg(f)/2^l$, which means that the degree of f would have to be very large in order for the guessing procedure to be unreliable. Hence this problem of having to be given information about the magnitude of $\deg(f)$ is more a theoretical than a practical issue.

Step 2: *Factor $f_2(X_1, X_2)$ over $\mathsf{K}[X_1, X_2]$ into irreducibles,*

$$f_2(X_1, X_2) =: \prod_{i=1}^{\tilde{r}} g_{2,i}(X_1, X_2)^{\tilde{e}_i}, \quad \tilde{e}_i \geq 1.$$

*An even simpler method of guessing $\deg(f_1)$ has been proposed by R. Liebling. Consider the following difference operator on $\mathsf{K}[X_1]$:

$$\Delta_0[\![f_1]\!] := f_1, \quad \Delta_{i+1}[\![f_1]\!](X_1) := \Delta_i[\![f_1]\!](X_1 + 1) - \Delta_i[\![f_1]\!](X_1) \text{ for all } i \geq 0.$$

For a random $A \in R$ we compute

$$\Delta_0[\![f_1]\!](A), \Delta_1[\![f_1]\!](A), \Delta_2[\![f_1]\!](A), \ldots$$

until we get $\Delta_{d+1}[\![f_1]\!](A) = 0$. Notice that a new $\Delta_{i+1}[\![f_1]\!](A)$ can be found by keeping the values of $\Delta_j[\![f_1]\!](A + i - j)$ with $0 \leq j \leq i$, and updating this fringe of a triangular difference table by the iteration: *For* $j \leftarrow 1, \ldots, i + 1$ *Do:*

$$\Delta_j[\![f_1]\!](A + i + 1 - j) \leftarrow \Delta_{j-1}[\![f_1]\!](A + i + 2 - j) - \Delta_{j-1}[\![f_1]\!](A + i + 1 - j).$$

Notice that this bivariate factorization problem is reducible to a univariate one, in polynomially many field operations (Kaltofen 1985a). Moreover, by virtue of the effective Hilbert irreducibility theorem (Kaltofen 1985b and 1989), for a set R of cardinality $2^{\deg(f)^{1+o(1)}}/\epsilon$ the factor degree patterns of f and f_2 agree with probability no less than $1-\epsilon$, that is, $d = \deg(f)$, $r = \tilde{r}$, and for all $1 \le i \le r$, $e_i = \tilde{e}_i$. Let us suppose that all this is true. Otherwise an incorrect output program will be produced.

Step 3: *Assign*

$$g_{1,i}(X_1) := g_{2,i}(X_1, 0) \quad \textit{for all } 1 \le i \le r.$$

Check whether $\mathrm{GCD}(g_{1,i}, g_{1,j}) = 1$ *for all* $1 \le i < j \le r$, *and whether* $\deg(g_{2,i}) = \deg(g_{1,i})$ *for all* $1 \le i \le r$. *If not, return "failure".* In that case we know that the random points were unlucky by zeroing one of several possible polynomials whose zeroes are to be avoided.

We now have, with high probability,

$$f_1(X_1) = f(X_1 + b_1, a_2X_1 + b_2, \ldots, a_nX_1 + b_n) = \prod_{i=1}^{r} g_{1,i}(X_1)^{e_i}, \tag{1}$$

where each $g_{1,i}$ is an image of an h_i. We first map from multi- to bivariate polynomials in step 2 and then from bi- to univariate polynomials in step 3 because of a lack of a univariate Hilbert irreducibility theorem for K. By that we mean a theorem that establishes the existence of a set of substitutions for the variables that map all irreducible factors of f to univariate irreducible polynomials. For general fields K such a theorem does not exist; e.g., if K is algebraically closed, f_1 will always factor into linear factors over K, even if f is irreducible. However, many other fields, such as finite algebraic extensions of $\mathbb{Q}$, yield a univariate Hilbert irreducibility theorem. Nonetheless, we know of no computationally effective versions, so for theoretical reasons we still must apply the bivariate counterpart. However, in practice it is unlikely that f violates the conclusion of such a theorem, which is that the factor degree pattern of f_1 with high probability agrees with that of f. Therefore, in practice, for such fields it is not unreasonable to map f directly to f_1 and then factor f_1 over $\mathsf{K}[X_1]$.

We will use $g_{1,i}$ to uniquely enumerate the factors of f. Our associate choices (see the output specifications above) will satisfy

$$h_i(X_1 + b_1, a_2X_1 + b_2, a_3X_1 + b_3, \ldots, a_nX_1 + b_n) = g_{1,i}(X_1).$$

Step 4: This step constructs the program for evaluation of the h_i at $p_1, \ldots, p_n$ as described in the output specifications. First, the information computed so far is 'hardwired' into that program. Then the following steps A, B, and C, are appended to the program.

Step A: *By standard interpolation using the determined* $\deg(f)$ *compute*

$$\begin{aligned}\bar{f}(X_1, Y) := f(X_1 + b_1, Y(p_2 - a_2(p_1 - b_1) - b_2) + a_2X_1 + b_2,\\ \ldots, Y(p_n - a_n(p_1 - b_1) - b_n) + a_nX_1 + b_n).\end{aligned}$$

Notice that

$$\bar{f}(p_1 - b_1, 1) = f(p_1, \ldots, p_n) \quad \text{and} \quad \bar{f}(X_1, 0) = f_1(X_1).$$

Step B: This step computes the factorization

$$\bar{f}(X_1, Y) = \prod_{i=1}^{r} \bar{g}_i(X, Y)^{e_i} \quad \text{with } \bar{g}_i(X_1, 0) = g_{1,i}(X_1) \text{ for all } 1 \le i \le r.$$

Note that since the polynomials $g_{1,i}$ are not necessarily irreducible, neither might be the $\bar{g}_i$. One way to obtain the factorization would be to factor $\bar{f}(X_1, Y)$ over $\mathsf{K}[X_1, Y]$ into irreducible factors, evaluate these factors at $Y = 0$, and then check which of the $g_{1,i}$ the corresponding univariate images divide. Finally, we can compute the product of those irreducible factors of $\bar{f}$ whose univariate images divide a given $g_{1,i}$ and adjust by a scalar multiplier to obtain $\bar{g}_i$. However, this would require the bivariate factorization of $\bar{f}$, and there is a much more efficient solution, namely by Hensel lifting the factorization (1) to one for $\bar{f}$. We will not describe the lifting algorithm in detail, but instead refer to (Musser 1976) and (von zur Gathen and Kaltofen 1985). The latter article explains that one can in our setting lift with the multiplicities e_i.
By Hensel lifting (1) obtain a factorization

$$\prod_{i=1}^{r} \bar{g}_i(X_1, Y)^{e_i} \equiv \bar{f}(X_1, Y) \pmod{Y^{d+1}}, \quad \deg_y(\bar{g}_i) \le d.$$

For all $1 \le i \le r$ test, whether $\bar{g}_i$ divides $\bar{f}$. If at least one test fails, return "program incorrect". In that case we have discovered that the factor degree patterns of f and f_2 must disagree.

Step C: *For $i \leftarrow 1, \ldots, r$ Do:*
Return $\bar{g}_i(p_1 - b_1, 1)$ as the value $h_i(p_1, \ldots, p_n)$, the value of the i-th irreducible factor.
□

The following two theorems discuss the complexity of this algorithm. As mentioned at the end of step 1, any failure estimates are dependent on the degree of f, which is not supplied as an input parameter and cannot be synthetically produced from the black box alone. Therefore, the failure probability can only be guaranteed if the set R in step 1 was selected of at least the stated cardinality.

THEOREM 1. *The Black Box Polynomial Factorization algorithm can construct its output program in polynomially many arithmetic steps as a function of n and $\deg(f)$ and an additional single polynomial factorization in $\mathsf{K}[X_1]$. It requires $O(\deg(f)^2)$ oracle calls to the black box for f. If the cardinality of the set R in step 1 is chosen*

$$\operatorname{card}(R) \ge 6 \deg(f)\, 2^{\deg(f)} / \epsilon,$$

then the algorithm succeeds with probability no less than $1 - \epsilon$ and the resulting program will always correctly evaluate all irreducible factors of f. That program in turn can be executed in polynomially many arithmetic steps and $O(\deg(f)^2)$ calls to the black box for f. For $\mathsf{K} = \mathbb{Q}(\vartheta)$, where ϑ is an algebraic number given by its minimal polynomial $\varphi(z) \in \mathbb{Q}[z]$, that is, K is isomorphic to $\mathbb{Q}[z]/(\varphi(z))$, the timings in terms of bit complexity are polynomial in n, $\deg(f)$, and the additional parameters: $\log(1/\epsilon)$, and the coefficient sizes of f and φ; this under the provision that the set R is chosen as the first $\operatorname{card}(R)$ positive integers. Furthermore, all evaluations of the black box for f within the algorithm are on input values of polynomial size as a function of the stated parameters.

Proof. The statements on the run time of both the algorithm and the returned program are easily verified. First, we need to interpolate the bivariate polynomials f_2 and $\bar{f}$, both of degree at most $\deg(f)$ (if unlucky elements are chosen in step 1, the degrees might be less). Clearly, each bivariate interpolation requires $O(\deg(f)^2)$ values of f. The algorithm then needs to factor f_2, which can be accomplished for $\mathsf{K} = \mathbb{Q}[z]/(\varphi(z))$ in binary polynomial time in $\deg(f_2)$ and the coefficient size of f_2 (Landau 1985). Notice that one needs to make an appropriate definition for the coefficient size of f in order to guarantee that the coefficient size of f_2 stays polynomially bounded. One such choice is the combined coefficient size of f as defined in Kaltofen (1989), §4, page 393. The dominating additional work of the output program is step B, which essentially is a uni- to bivariate lifting problem that certainly can be accomplished in $O(\deg(f)^4)$ arithmetic operations.

It remains to analyze the failure probabilities. An incorrect program is output if and only if the factor degree patterns of f and f_2 disagree. This happens when $\deg(f_2) < \deg(f)$, or when some h_i maps to a reducible polynomial in $\mathsf{K}[X_1, X_2]$, or when two of the h_i's map to associated irreducible polynomials in $\mathsf{K}[X_1, X_2]$. By theorem 5.2 in (Kaltofen 1989) (or as explained in the proof of theorem 6.1 (ibid.)), this happens with probability less than

$$(\deg(f) + 4\deg(f)\, 2^{\deg(f)} + \deg(f)^3)/\mathrm{card}(R).$$

Furthermore, if we use the procedure described in step 1 for guessing the degree of f, we may pick a value A that leads to an agreement $f_1(A) = f_1^{(d)}(A)$ for $d < \deg(f)$ with probability no more than $\deg(f)^2/\mathrm{card}(R)$. Thus, by choosing $\mathrm{card}(R) > 6\deg(f)\, 2^{\deg(f)}/\epsilon$ we can guarantee that the output program is correct (including that step B will never report an error) with probability no less than $1 - \epsilon$.

Finally, we need to estimate the probability that step 3 reports failure. Such circumstances correspond to failure in step F of the Factorization algorithm in (Kaltofen 1989), and is treated in the fourth case of failure in the proof of theorem 6.1 (ibid.). It is shown that the probability of these events can be bounded from above by $\deg(f)^2/\mathrm{card}(R)$, which for the given cardinality of R certainly lies below ϵ. ⊠

Theorem 1 has several applications to computing the sparse factorization of a multivariate polynomial given by a black box. For instance, we get the following theorem of theoretical interest, which establishes that several sparse factorization questions as belonging to the complexity class randomized $\mathcal{NC}$ for parallel computation (see (Cook 1985)); it also solves an open problem in (von zur Gathen and Kaltofen 1985).

THEOREM 2. *For $\mathsf{K} = \mathbb{Q}(\vartheta)$ as in theorem 1, we have a Monte Carlo $\mathcal{NC}$-reduction from the problem of computing all sparse factors with no more than t terms to the problem of factoring univariate polynomials over $\mathbb{Q}$. This reduction is of polynomial binary operation count and poly-logarithmic depth in terms of n, $\deg(f)$, t, and the coefficient size of f and φ. It requires that the black box calls account for polynomially many Boolean operations and poly-logarithmic depth. In particular, under this assumption for $\mathsf{K} = \mathbb{Q}$ the problem of computing all sparse factors of a fixed degree with no more than t terms is in the complexity class Monte Carlo $\mathcal{NC}$.*

Proof. Constructing and evaluating the output program of the Black Box Polynomial Factorization algorithm is basically interpolating bivariate images of the input black box and bivariate factorization. Bivariate factorization over $\mathsf{K} = \mathbb{Q}[z]/(\varphi(z))$ is $\mathcal{NC}$ reducible to univariate factorization over $\mathbb{Q}$ by the reduction from bivariate to univariate factorization over K in (Kaltofen 1985a) and the reduction from factoring univariate polynomials over algebraic number fields to factoring univariate polynomials over the rationals in (Trager 1976). From the program for all factors we can deduce the sparse factors in Monte Carlo $\mathcal{NC}$ by the algorithm of Ben-Or and Tiwari (1988). Note that although the Ben-Or and Tiwari algorithm is deterministic, we must check which of the sparse interpolating polynomials with t or fewer non-zero monomials are correct because the Ben-Or and Tiwari algorithm can yield an incorrect sparse interpolating polynomial for a dense factor. The verification can be done in a Monte Carlo fashion by evaluating both the program for the factor and the computed sparse candidate polynomial at a random point and by comparing the resulting algebraic numbers.

We finally discuss the statement on factors of fixed degree. In step 2 one can in parallel compute those bivariate integral factors that have fixed degree by lifting univariate factors modulo a prime number first to integral factors and then to bivariate factors as in the standard Berlekamp-Hensel method (Kaltofen 1982). The modular univariate factors can be found in parallel by a variation of the Berlekamp algorithm (Gathen 1984). Finally, both the true bivariate factors as well as the true univariate integral factors are recoverable within the complexity $\mathcal{NC}$ because there are only a polynomial number of lifted factor combinations to be tested, since the degree of the product candidate polynomial is fixed. ⊠

We shall conclude this section with a different interpretation of our idea, which was actually conceived after the method had been discovered. One can view steps B and step C in the algorithm to fit the homotopy continuation paradigm for numerically solving a system of equations (see, for instance, (Drexler 1977), (Garcia and Zangwill 1979), (Li, Sauer, and Yorke 1988), and (Zulehner 1988)). Assume that one wants to solve the system of equations $\vec{F}(\vec{x}) = 0$. One parameterizes the indeterminates with a single new variable Y and considers the system

$$\vec{H}(\vec{X}(Y)) = (1 - Y)\vec{G}(\vec{X}(Y)) + Y\vec{F}(\vec{X}(Y)),$$

where $\vec{X}(0)$ is a solution to the specially selected system $\vec{G}(\vec{x}) = 0$ and $\vec{X}(1)$ is sought as a solution of $\vec{H}(\vec{X}(1)) = 0$. In order to obtain $\vec{X}(1)$ one continously deforms $\vec{X}(0)$ by letting the parameter Y go from 0 to 1. Note that in the numerical homotopy methods, the standard parameterization described for the variables is simply $\vec{X}(Y) = \vec{x}$, that is the systems $\vec{G}$ and $\vec{F}$ stay invariant during the deformation.

In our case the problem $\vec{G}(\vec{x}) = 0$ is the problem of factoring f_1, and the problem $\vec{F}(\vec{x}) = 0$ is the problem of finding the corresponding factorization of $\bar{f}(X_1, 1)$. The former problem does not depend on the input $p_1, \ldots, p_n$, whereas the latter does. The Hensel lifting method referred to in step B exactly corresponds to deforming the solution of $\vec{G}(\vec{x}) = 0$ to one for $\vec{H}(\vec{X}(Y))$, although it should be noted that our deformation deforms with a discrete valuation. Since $\deg_Y(\bar{f})$ is finite, this deformation terminates with the exact solution, and evaluation at $Y = 1$ is possible.

3. Black Box GCDs and Separation of Numerators from Denominators

We now discuss two more results concerning the construction of evaluation programs from black boxes. The first is the construction of a program for the evaluation of the greatest common divisor of multivariate polynomials, the second takes a black box for a multivariate rational function and will individually evaluate its numerator and denominator. As in the factorization algorithm, the resulting programs are, with high probability, universally correct: they will return the correct values for all inputs. This requirement makes the construction more difficult. In both the GCD and the separation result we will employ the same idea that we already used in the factorization algorithm above.

We first discuss the computation of GCDs and shall give only a brief explanation. Assume that for $i = 1, \ldots, r$, $r \geq 2$, the non-zero polynomials $f_i(x_1, \ldots, x_n) \in \mathsf{K}[x_1, \ldots, x_n]$ are given by a black box. For field elements $p_1, \ldots, p_n$ we wish to have $g(p_1, \ldots, p_n)$, where $g = \mathrm{GCD}(f_1, \ldots, f_r)$ and we again fix the associate. First we choose random field elements

$$a_2, \ldots, a_n, b_2, \ldots, b_n, c_2, \ldots, c_r.$$

Now with high probability for the univariate image of the GCD,

$$g_1(X) := g(X, a_2X + b_2, \ldots, a_nX + b_n) \in \mathsf{K}[X],$$

we have $\deg(g_1) = \deg(g)$, that is we can normalize g such that g_1 is monic. Furthermore, with high probability (see (Kaltofen 1988, Theorem 6.2)), we have

$$g_1 = \mathrm{GCD}(f_1(X, a_2X + b_2, \ldots, a_nX + b_n), \sum_{i=2}^{r} c_i f_i(X, a_2X + b_2, \ldots, a_nX + b_n)).$$

By polynomial interpolation we now compute the two bivariate polynomials

$$\bar{f}_0(X, Y) := \sum_{i=2}^{r} c_i \bar{f}_i(X, Y) \quad \text{and} \quad \bar{f}_1(X, Y),$$

where the $\bar{\ }$ operator is for any $h \in \mathsf{K}[x_1, \ldots, x_n]$ the projection

$$\bar{h}(X, Y) := h(X, Y(p_2 - a_2p_1 - b_2) + a_2X + b_2, \ldots, Y(p_n - a_np_1 - b_n) + a_nX + b_n).$$

Next, we compute $\bar{g}(X, Y)$ as the $\mathrm{GCD}(\bar{f}_0(X, Y), \bar{f}_1(X, Y))$, where $\bar{g}$ is normalized by making the leading coefficient $\mathrm{ldcf}_X(\bar{g}) = 1$. We actually get $\bar{g}$ and the normalization amounts to a scalar division if g_1 does not drop in degree compared with the degree of g, that with high probability. Finally, we return as $g(p_1, \ldots, p_n)$ the value $\bar{g}(p_1, 1)$.

Even though this is more or less the same idea as the one in the factorization algorithm, the transcendental Y is used for a somewhat different reason here. The problem is that one cannot guarantee that the GCD of the polynomials $\bar{f}_0(X, 1)$ and $\bar{f}_1(X, 1)$ is an image of g, and our algorithm is to produce a black box which is correct with high probability; correct means that it evaluates the GCD at all points. It is therefore intriguing to compare the result with the polynomial GCD algorithm for the straight-line representation (Kaltofen 1988, §6). In fact, a similar problem arises for the solution presented there, but in a somewhat more implicit way. Although the resulting straight-line program for the GCD is derived by a univariate polynomial remainder sequence (see step R in the cited algorithm), that program will contain divisions. Therefore, it cannot be evaluated at all points in the straight-forward manner. One has to first remove the divisions by Strassen's algorithm (see Theorem 7.1 in (Kaltofen 1988)). That again leads a consequent increase in the complexity for computing the values everywhere.

As the last Black Box result in this article we come to the problem of computing the values of the numerator and the denominator of a rational function. A certain subproblem, that of a mixed radix Padé approximation, will be used in both the algorithm constructing the resulting program and in the program itself. For clarity, we shall present the algorithm for this subproblem below the main algorithm.

Algorithm *Black Box Numerator and Denominator*

Input: A black box that evaluates $f(x_1, \ldots, x_n)/g(x_1, \ldots, x_n) \in \mathsf{K}(x_1, \ldots, x_n)$, where K is a field of characteristic 0, $f, g \in \mathsf{K}[x_1, \ldots, x_n]$ with $\mathrm{GCD}(f, g) = 1$. The black box returns ∞ if one evaluates at a root of g. Furthermore a failure probability $\epsilon \ll 1$ is input. Without the knowledge of a bound $\bar{e} \geq \deg(g)$, this algorithm might with positive probability loop forever. If this is not desired, a degree bound $\bar{e}$ is also required as input.

Output: A program (see Figure 4) that makes oracle calls to the black box for f/g and has with probability at least $1 - \epsilon$ the following property. The program accepts n field elements $p_1, \ldots, p_n$. It returns the values of

$$f(p_1, \ldots, p_n) \quad \text{and} \quad g(p_1, \ldots, p_n).$$

As in the Black Box Polynomial Factorization algorithm, certain associates for f and g are chosen once and for all and the values of these associates will be returned.

Notice that the program will be able to return the value of $f(p_1, \ldots, p_n)$ even if $g(p_1, \ldots, p_n) = 0$. This apparent paradox—the black box for f/g does not supply any information on f when probed at $p_1, \ldots, p_n$—is resolved by deducing the value of $f(p_1, \ldots, p_n)$ from values at other points that are not zeros of g.

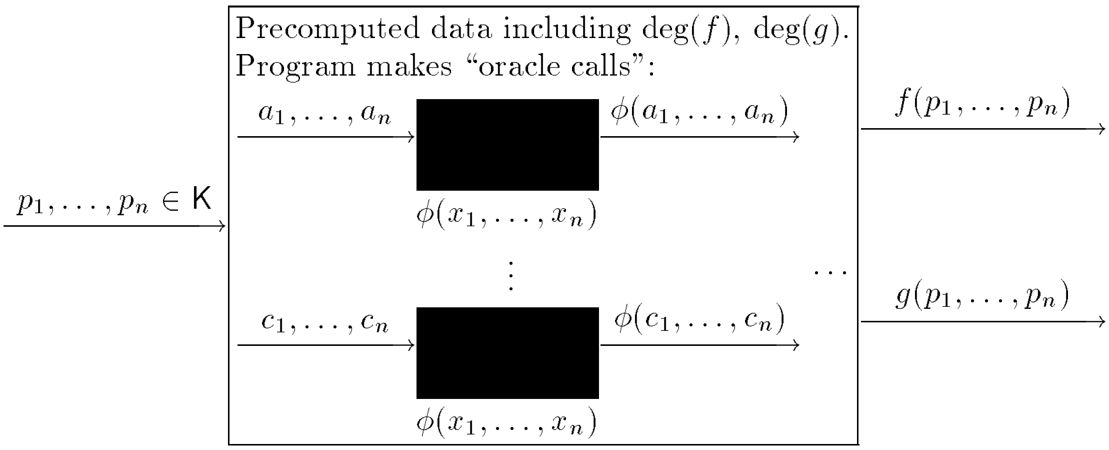
Step 1: This step introduces a random projection of f and g into $\mathsf{K}[X]$ that with high probability preserves their relative primality and the degree of g.
From a sufficiently large finite set $R \subset \mathsf{K}$ randomly select a_i, b_i, $2 \leq i \leq n$. In place of x_i we will evaluate at $a_i X + b_i$ for $2 \leq i \leq n$, such that for

$$f_1(X) := f(X, a_2 X + b_2, \ldots, a_n X + b_n)$$

and

$$g_1(X) := g(X, a_2 X + b_2, \ldots, a_n X + b_n)$$

we have $\mathrm{GCD}(f_1, g_1) = 1$ over $\mathsf{K}[X]$ and $\deg(g_1) = \deg(g)$: depending on the cardinality of R this is true with high probability. Notice that the degree condition on g_1 implies that $g_1 \neq 0$.



$$\phi(x_1, \ldots, x_n) = \frac{f(x_1, \ldots, x_n)}{g(x_1, \ldots, x_n)}, f, g \in \mathsf{K}[x_1, \ldots, x_n], \mathrm{GCD}(f, g) = 1.$$

Figure 4: The program for evaluating the numerator and denominator of a black box rational function.

Step 2: This step probabilistically computes the degrees of f and g. In order to prevent an infinite loop in the case that $g_1 = 0$ we assume that one is given an additional bound $\bar{e} \geq \deg(g)$.

For $\delta \leftarrow 0, 1, 2, \ldots$ Do:

Call the algorithm Special Padé Approximation below with the degree bounds $d = e = \delta$ and the linear substitutes $a_2X + b_2, \ldots, a_nX + b_n$. Denote the two obtained polynomials by $f_1^{(\delta)}, g_1^{(\delta)} \in \mathsf{K}[X]$. If both polynomials are 0, then either the projection sent g into the zero polynomial or $\delta < \deg(g)$ together with additional conditions on g_1 (see the output specifications of the Special Padé Approximation Algorithm below).

If $f_1^{(\delta)} = g_1^{(\delta)} = 0$ increment δ by one and continue the For loop, unless $\delta = \bar{e}$, in which case return to step 1. In this exceptional case $g_1 = 0$, provided that $\bar{e}$ is a true bound for $\deg(g)$. Therefore a new random projection must be tried.

If $g_1^{(\delta)} \neq 0$, pick another random element $A \in R$ and compare $(f/g)(A, a_2A + b_2, \ldots, a_nA + b_n)$ computed by the input black box with the value of $f_1^{(\delta)}(A)/g_1^{(\delta)}(A)$. If both values are the same and are not ∞ then declare $\deg(f)$ to be $\deg(f_1^{(\delta)}) \to d$ and $\deg(g)$ to be $\deg(g_1^{(\delta)}) \to e$, and go to step 3. If both values are ∞, pick a new distinct random A and repeat the comparison. Otherwise continue the For loop with the next δ.

Step 3: This step constructs the program described in the output specifications. It will permanently use the random elements chosen in step 1 and the degrees computed in step 2. The idea is to compute

$$\bar{f}(X, Y) := f(X, a_2X + b_2 + Y(p_2 - a_2p_1 - b_1), \ldots, a_nX + b_n + Y(p_n - a_np_1 - b_n))$$

and

$$\bar{g}(X, Y) := g(X, a_2X + b_2 + Y(p_2 - a_2p_1 - b_1), \ldots, a_nX + b_n + Y(p_n - a_np_1 - b_n)).$$

For $Y = 0$ we already have computed these polynomials in step 2. Here we compute $\bar{f}_j(X) := \bar{f}(X, i_j)$ and $\bar{g}_j(X) := \bar{g}(X, i_j)$ for sufficiently many distinct integral values i_j, that we can interpolate with respect to Y. However, in order to use the special Padé approximation for the computation of $\bar{f}_j$ and $\bar{g}_j$ these two images must be relatively prime. Therefore, certain values for i_j have to be discarded.

Step A: $j \leftarrow 0$.
For $I \leftarrow 1, 2, \ldots$ *Do:*

Call the algorithm Special Padé Approximation below with the degree bounds d, e *and the linear substitutions* $a_i X + b_i + I(p_i - a_i p_1 - b_i)$ *for* $2 \le i \le n$. *If the degrees of the returned polynomials are below* d *and* e *respectively, continue the For loop with the next* I. *Otherwise, set* $i_{j+1} \leftarrow I$, $\bar{f}_{j+1}(X)$ *to the numerator polynomial returned, and* $\bar{g}_{j+1}(X)$ *to the denominator polynomial returned. Increment* j *by* 1. *If* $j = \max(d, e) + 1$ *then exit the loop. Otherwise continue the For loop with the next* I.

As we will show below, if the substitutions of step 1 are lucky the above loop will discard at most $2de$ values of I. Thus, if more values are unusable the entire program is incorrect. Of course, this should be tested for and diagnosed, instead of getting into a possible infinite loop in such an unlucky case.

Step B: *From the polynomials* $\bar{f}_j, \bar{g}_j$ *compute, by interpolation, candidates for the polynomials* $\bar{f}(X, Y), \bar{g}(X, Y)$.

Step C: *Return* $\bar{f}(p_1, 1)$ *and* $\bar{g}(p_1, 1)$ *as the values of* $f(p_1, \ldots, p_n)$ *and* $g(p_1, \ldots, p_n)$ *respectively.* □

Algorithm *Special Padé Approximation*

Input: The same black box for f/g that is input of the Numerator and Denominator algorithm. Furthermore, two degree bounds d, e and linear forms $u_2 X + v_2, \ldots, u_n X + v_n$ with $u_i, v_i \in \mathsf{K}$ for $2 \le i \le n$.

Output: Two polynomials $\hat{f}(X), \hat{g}(X) \in \mathsf{K}[X]$ with the following property. The algorithm has chosen $d + e + 1$ integers

$$1 \le i_1 < i_2 < \cdots < i_{d+e+1}$$

such that for all $1 \le k \le d + e + 1$,

$$\frac{\hat{f}(i_k)}{\hat{g}(i_k)} = \frac{f}{g}(i_k, u_2 i_k + v_2, \ldots, u_n i_k + v_n), \tag{2}$$

and such that

$$\deg(\hat{f}) \le d, \quad \deg(\hat{g}) \le e, \quad \mathrm{ldcf}(\hat{g}) = 1.$$

If the right-hand side of (2) is not ∞ for all k, then such $\hat{f}, \hat{g}$ can be determined (see lemma 1 below). Furthermore, if $d \ge \deg(f)$ and $e \ge \deg(g)$, the algorithm will return relatively prime polynomials $\hat{f}, \hat{g}$. (Refer also to the remark following the algorithm description.) No appropriate i_k can be selected if either

$$0 = g(X, u_2 X + v_2, \ldots, u_n X + v_n) =: \tilde{g}(X),$$

or if $\deg(g) > e$ and more than e of the integers j in $1 \le j \le d + 2e + 1$ are roots of $\tilde{g}(X)$. In these exceptional cases two zero polynomials are returned.

Step I: We first find, if possible, a set of $d + e + 1$ integers $1 \leq i_1 < i_2 < \cdots < i_{d+e+1} \leq d + 2e + 1$ that are not roots of $\tilde{g}(X) = g(X, u_2X + v_2, \ldots, u_nX + v_n) \in \mathsf{K}[X]$.
$k \leftarrow 0$.
For $J \leftarrow 1, \ldots, d + 2e + 1$ *Do:*
By evaluation of the black box for f/g *compute*

$$A_{k+1} \leftarrow \frac{f}{g}(J, u_2J + v_2, \ldots, u_nJ + v_n).$$

If $A_{k+1} \neq \infty$ *then* $i_{k+1} \leftarrow J$. *Increment* k *by* 1 *and if* $k = d + e + 1$ *exit loop.*
At this point we have tried at most $d + 2e + 1$ values for J.
If $k < d + e + 1$ *then return two zero polynomials for* $\hat{f}$ *and* $\hat{g}$. With

$$\tilde{f}(X) := f(X, u_2X + v_2, \ldots, u_nX + v_n)$$

we now have $A_k = \tilde{f}(i_k)/\tilde{g}(i_k)$ for all $1 \leq k \leq d + e + 1$.

Step II: *By polynomial interpolation compute a polynomial* $h(X) \in \mathsf{K}[X]$ *with* $h(i_k) = A_k$ *for all* $1 \leq k \leq d + e + 1$ *and* $\deg(h) \leq d + e$.

Step III: Now we compute polynomials $\hat{f}(X), \hat{g}(X) \in \mathsf{K}[X]$ such that

$$\hat{f}(X) \equiv \hat{g}(X)h(X) \pmod{(X - i_1) \cdots (X - i_{d+e+1})},$$

and

$$\mathrm{GCD}(\hat{f}, \hat{g}) = 1, \quad \deg(\hat{f}) \leq d, \quad \deg(\hat{g}) \leq e, \quad \mathrm{ldcf}(\hat{g}) = 1.$$

This is accomplished by the extended Euclidean algorithm as follows:
Initialize

$$\begin{array}{ll} h_0(X) \leftarrow (X - i_1) \cdots (X - i_{d+e+1}); & g_0(X) \leftarrow 0; \\ h_1(X) \leftarrow h(X); & g_1(X) \leftarrow 1. \end{array}$$

For $l \leftarrow 1, 2, \ldots$ *Do:*
If $\deg(h_l) \leq d$ *then exit the loop. Otherwise, perform a polynomial division with remainder of* h_{l-1} *and* h_l,

$$h_{l+1}(X) \leftarrow h_{l-1}(X) \pmod{h_l(X)};$$
$$q_{l+1}(X) \leftarrow \big(h_{l-1}(X) - h_{l+1}(X)\big)/h_l(X);$$
$$g_{l+1}(X) \leftarrow g_{l-1}(X) - q_{l+1}(X)g_l(X).$$

Return $\hat{f} \leftarrow h_l/\mathrm{ldcf}(g_l); \hat{g} \leftarrow g_l/\mathrm{ldcf}(g_l)$. □

We first discuss the auxiliary Special Padé Approximation algorithm in more detail. It can be shown that in the case where $d < \deg(f)$ or $e < \deg(g)$ the fraction $\hat{f}/\hat{g}$ of polynomials satisfying the output conditions is uniquely determined (see, for instance, (Knuth 1981, Exercise 4.6.1–26)). However, since we want to avoid the GCD reduction to relative prime outputs at the end of the algorithm, and because we do not need this fact in step 2, we will not provide a proof. Nonetheless, all other needed conditions can be deduced from the following simple lemma, whose content is long known (see, for instance, (Gragg 1972)).

LEMMA 1. *Let d and e be non-negative integers, and let $F(X), G(X), H(X) \in \mathsf{K}[X]$, K an arbitrary field, $\deg(H) < d+e+1$, $\mathrm{GCD}(F, G) = 1$; furthermore, let i_k, $1 \le k \le d+e+1$, be not necessarily distinct elements in K such that*

$$F \equiv G\,H \pmod{(X - i_1)\cdots(X - i_{d+e+1})}.$$

Define $h_0(X) := (X - i_1)\cdots(X - i_{d+e+1})$, $\delta_0 := d+e+1$, and $h_1(X) := H(X)$, $\delta_1 := \deg(H)$. Now let $h_l(X), q_l(X) \in \mathsf{K}[X]$ be the l-th remainders and quotients respectively, in the Euclidean polynomial remainder sequence

$$h_{l-2}(X) = q_l(X)h_{l-1}(X) + h_l(X), \quad \delta_l := \deg(h_l) < \delta_{l-1} \text{ for } l \ge 2.$$

In the exceptional case $H = 0$ the sequence is defined to be empty.
Finally, let $f_l(X), g_l(X) \in \mathsf{K}[X]$ be the multipliers in the extended Euclidean scheme $f_l h_0 + g_l h_1 = h_l$, namely,

$$f_0 := g_1 := 1, \quad f_1 := g_0 := 0,$$
$$f_l := f_{l-2} - q_l f_{l-1}, \quad g_l := g_{l-2} - q_l g_{l-1} \quad \text{for } l \ge 2.$$

Then there exists an index j, $1 \le j$, such that $\delta_j \le d < \delta_{j-1}$. For that index we have

$$h_j \equiv g_j H \pmod{(X - i_1)\cdots(X - i_{d+e+1})} \quad \text{and} \quad \deg(g_j) \le e. \tag{3}$$

Furthermore, if $d \ge \deg(F)$ and $e \ge \deg(G)$ then $F = \lambda h_j$, $G = \lambda g_j$ for some $\lambda \in \mathsf{K}$.

Proof. It follows by induction on l that

$$\forall\, l \ge 2\colon f_l h_0 + g_l h_1 = h_l, \quad \deg(f_l) = \delta_1 - \delta_{l-1}, \quad \deg(g_l) = \delta_0 - \delta_{l-1}.$$

Furthermore, one can easily show that $f_l g_{l-1} - f_{l-1} g_l = \pm 1$, which implies that $\mathrm{GCD}(f_l, g_l) = 1$ (see (Knuth 1981, §4.6.1, Exercise 3)). Since in the Euclidean sequence the remainder 0 eventually appears, an index j can always be found. Now, we have

$$h_j \equiv g_j H \pmod{h_0}, \quad \deg(g_j) = \delta_0 - \delta_{j-1} \le e.$$

This shows that the pair of polynomials h_j, g_j is a solution to (3).
We finally prove uniqueness in case $d \ge \deg(F)$ and $e \ge \deg(G)$. Observing that

$$g_j F - h_j G = (F - GH)\,g_j - (h_j - g_j H)\,G \equiv 0 \pmod{h_0},$$

and the left hand side polynomial has degree less than $d+e+1$, we must have $g_j F = h_j G$. Thus there is a $w(x) \in \mathsf{K}[x]$ with $F = h_j/w$, $G = g_j/w$. Plugging into $h_j = f_j h_0 + g_j h_1$ we infer that then

$$\frac{f_j h_0}{w} = F - GH \equiv 0 \pmod{h_0},$$

which implies that w must divide f_j. From $\mathrm{GCD}(f_j, g_j) = 1$ we thus conclude that $w \in \mathsf{K}$. ⊠

The analysis of the above algorithms now leads us to a theorem very similar in spirit to theorem 1. Notice that only the cardinality of R depends on the failure probability bound ϵ, thus making the arithmetic complexity independent of it, while for properly chosen R (see theorem 1) the binary complexity is polynomially dependent on $\log(1/\epsilon)$ as well. Again as for theorem 1, bounds for the degrees of f and g are required in the estimates. Without any knowledge of degree bounds, one can neither guarantee termination nor estimate the failure probability in any way, since the black box for might evaluate a rational function of astronomical degree for which all choices taken from a selected set R are bad.

THEOREM 3. *Given a bound $\bar{e} \geq \deg(g)$ as an additional input, the algorithm Black Box Numerator and Denominator can construct its output program in polynomially many arithmetic steps as a function of n, $\deg(f)$ and $\bar{e}$. It requires $O((\deg(f)+\bar{e})^2)$ oracle calls to the black box for f/g. If the cardinality of the set R in step 1 is chosen at least*

$$\max\Bigl(2\,(2\deg(f)+1)\deg(g), 3m^2-m\Bigr)\Big/\epsilon, \quad m := \max(\deg(f), \deg(g)),$$

then the resulting program correctly evaluates f and g with probability no less than $1-\epsilon$. That program in turn can be executed in polynomially many arithmetic steps and it makes $O(\deg(f)\deg(g)(\deg(f)+\deg(g)))$ calls to the black box for f/g.

Proof. The Special Padé Approximation algorithm takes $O(d+e)$ evaluations of the input black box, where d and e are the supplied degrees. Clearly, step 2 will call this algorithm at most $\max(\deg(f), \bar{e})$ times until the original and the computed rational functions agree at A. In order to deduce the number of calls to the Padé algorithm in step A, we need to estimate how many I can lead to lower degrees. The argument is fairly standard. Consider the Sylvester resultant (see (van der Waerden 1953, §7))

$$\rho_1(Y) := \mathrm{Res}_X(\bar{f}(X,Y), \bar{g}(X,Y)).$$

If f_1 and g_1 are relatively prime images of f and g with $\deg(g_1) = \deg(g)$, then

$$\rho_1(0) = \mathrm{Res}_X(f_1(X), g_1(X)) \neq 0.$$

Furthermore, $\deg(\rho_1) \leq 2de$, and any I with $\rho_1(I) \neq 0$ will satisfy $\mathrm{GCD}(\bar{f}(X,I), \bar{g}(X,I)) = 1$, which proves that in the lucky case at most $2de$ of the I are unusable. Hence the number of evaluations of the input black box that are triggered in step A are at most

$$(2de + \max(d,e) + 1)(d + 2e + 1) = O(\deg(f)\deg(g)(\deg(f)+\deg(g))).$$

It remains to estimate the probability that the returned program is incorrect. Again the argument is standard. Consider the generic leading coefficient

$$g(X, \alpha_2 X + \beta_2, \ldots, \alpha_n X + \beta_n)) =: \tau_1(\alpha_2, \ldots, \alpha_n)\, X^{\deg(g)} + \cdots,$$

and consider the Sylvester resultant

$$\rho_2(\alpha_2, \ldots, \beta_n) := \mathrm{Res}_X(f(X, \alpha_2 X + \beta_2, \ldots, \alpha_n X + \beta_n), g(X, \alpha_2 X + \beta_2, \ldots, \alpha_n X + \beta_n)).$$

Clearly, $\tau_1(a_2, \ldots, a_n)\rho_2(a_2, \ldots, b_n) \neq 0$ implies $\mathrm{GCD}(f_1, g_1) = 1$ and $\deg(g_1) = \deg(g)$. Since $\deg(\tau_1\rho_2) \leq (2\deg(f)+1)\deg(g)$, the probability of missing this condition in step 1 is no more than

$$\frac{(2\deg(f)+1)\deg(g)}{\mathrm{card}(R)} \leq \frac{\epsilon}{2}.$$

Finally, step 2 must return the correct degrees under the assumption that step 1 picked a lucky evaluation. An element A establishes too low a degree if, despite the fact that $\delta < m = \max(\deg(f), \deg(g))$, it is a zero of the polynomial

$$\tau_2(X) := f_1(X)\, g_1^{(\delta)}(X) - g_1(X)\, f_1^{(\delta)}(X) \neq 0.$$

However, $\deg(\tau_2) \leq m + \delta$, so A will be a witness for all $\delta < m$ with probability at least

$$1 - \frac{3m^2 - m}{2\,\mathrm{card}(R)} \geq 1 - \frac{\epsilon}{2}. \quad \boxtimes$$

As for theorem 1, when K is an algebraic number field and in particular the rational numbers, the algorithm Black Box Numerator and Denominator is of polynomial-time bit complexity under the usual assumptions. The algorithm also can be used to solve the sparse rational function recovery problem (Ben-Or and Tiwari 1988). One version of that problem assumes that one has term count as well as degree bounds for both the numerator and the denominator. Unlike the sparse polynomial interpolation algorithm in (Ben-Or and Tiwari 1988), our solution uses randomization. However, the final answer can be verified, that is the randomization can be made 'Las Vegas'. Let t be a common term bound for both the actual numerator f and denominator g, and let $\tilde{f}$ and $\tilde{g}$ be the computed sparse numerator and denominator polynomials, respectively. Notice that the sparse interpolation methods might return incorrect polynomials, but they will always have no more than t terms. Therefore $h := f\tilde{g} - \tilde{f}g$ is a polynomial with no more than $2t^2$ terms. Then, if

$$\frac{\tilde{f}(p_1^l, \ldots, p_n^l)}{\tilde{g}(p_1^l, \ldots, p_n^l)} = \frac{f}{g}(p_1^l, \ldots, p_n^l)$$

for distinct primes $p_1, \ldots, p_n$ and for all l with $1 \leq l \leq 2t^2$, then we must have $h = 0$ (Zippel 1990), that is $\tilde{f} = f$ and $\tilde{g} = g$. We have the following theorem.

THEOREM 4. *Given a black box for f/g as in the algorithm Black Box Numerator and Denominator, and given an upper bound t for the number of non-zero monomials in both f and g, we can in expected polynomially many arithmetic steps in $\deg(f)$, $\deg(g)$, n, and t compute the sparse representations of f and g.*

Proof. One distinction of this to the previous results is that no degree estimates are needed as inputs. The reason is that the verification procedure discussed above is always correct. Therefore, the algorithm can step through degree estimates and select a set R such that the failure probability is exponentially small. This strategy may lead to an endless run, but the expected value of the run time will be polynomial in the said parameters. ⊠

Notice that this theorem can also be formulated in terms of bit complexity for an algebraic coefficient field $\mathsf{K} = \mathbb{Q}(\vartheta)$ similar to treatment of this special case in theorem 1.

4. Conclusion

Oracle calls in algorithms is a ubiquitous paradigm in complexity reductions such as Turing reductions in recursive function theory, Cook reductions in the theory of $\mathcal{NP}$-completeness, or reductions that relate the matrix multiplication complexity asymptotically to other linear algebra problems such as solving linear systems. Here we have introduced this paradigm to several classical problems on multivariate polynomials. We solved the greatest common divisor, factorization, and numerator/denominator problems.

One application of the black box factorization procedure is that to factor u-resultants (Macaulay 1916). In fact, in that case all factors are known to be linear forms and the algorithm specializes to a method very similar to that in (Canny 1988). An efficient black box for evaluating the u-resultant is discussed in (Canny et al. 1989), and this algorithm, based on Wiedemann's (1986) randomized sparse determinant procedure, is particularly hard to formulate in the straight-line model.

From a pragmatic point of view, the black box model appears to be superior to the straight-line program model because of space requirements. Algorithms acting on straight-line programs usually amplify the length of the result by a factor depending on the degree, whereas the programs making oracle black box calls are of small additional size over the input black boxes. Furthermore, one can adapt most of the results to finite coefficient fields of sufficiently high characteristic and thus avoid costly rational coefficient arithmetic. Moreover, the resulting evaluation programs are especially easy to distribute to asynchronous parallel processors for multiple evaluation in parallel, and perhaps subsequent sparse interpolation. We believe that the black box approach not only provides a simple, but also a practically very efficient solution to, say, the problem of sparse multivariate polynomial factorization.

Acknowledgement: We thank Bobby Caviness, Stephen Cook, Lakshman Yagati, and Prasoon Tiwari for their comments on this work. We especially thank John A. Abbott, David Bayer, and one anonymous referee for their detailed comments to this paper.

Literature Cited

Baur, W. and Strassen, V., "The complexity of partial derivatives," *Theoretical Comp. Sci.* **22**, pp. 317–330 (1983).

Ben-Or, M. and Tiwari, P., "A deterministic algorithm for sparse multivariate polynomial interpolation," *Proc. 20th Annual ACM Symp. Theory Comp.*, pp. 301–309 (1988).

Canny, J., *The Complexity of Robot Motion Planning*; ACM Ph.D. Thesis Award 1987; MIT Press, Cambridge, MA, 1988.

Canny, J., Kaltofen, E., and Lakshman, Yagati, "Solving systems of non-linear polynomial equations faster," *Proc. ACM-SIGSAM 1989 Internat. Symp. Symbolic Algebraic Comput.*, pp. 121–128 (1989).

Cook, S. A., "A taxonomy of problems with fast parallel algorithms," *Inf. Control* **64**, pp. 2–22 (1985).

Drexler, F. J., "Eine Methode zur Berechnung sämtlicher Lösungen von Polynomgleichungssystemen," *Numer. Math.* **29**, pp. 45–58 (1977). (In German.)

Freeman, T. S., Imirzian, G., Kaltofen, E., and Lakshman, Yagati, "DAGWOOD: A system for manipulating polynomials given by straight-line programs," *ACM Trans. Math. Software* **14**/3, pp. 218–240 (1988).

Garcia, C. B. and Zangwill, W. I., "Finding all solutions to polynomial systems and other systems of equations," *Math. Program.* **16**, pp. 159–176 (1979).

von zur Gathen, J., "Parallel algorithms for algebraic problems," *SIAM J. Comp.* **13**, pp. 802–824 (1984).

von zur Gathen, J., "Irreducibility of multivariate polynomials," *J. Comp. System Sci.* **31**, pp. 225–264 (1985).

von zur Gathen, J., "Representations and parallel computations for rational functions," *SIAM J. Comp.* **15**, pp. 432–452 (1986).

von zur Gathen, J. and Kaltofen, E., "Factoring sparse multivariate polynomials," *J. Comp. System Sci.* **31**, pp. 265–287 (1985).

Golub, G. H. and van Loan, C. F., *Matrix Computations*; Johns Hopkins University Press, Baltimore, Maryland, 1987.

Gragg, W. B., "The Padé table and its relation to certain algorithms of numerical analysis," *SIAM Review* **14**/1, pp. 1–62 (1972).

Grigoryev, D. Yu., Karpinski, M., and Singer, M. F., "Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields," *SIAM J. Comput.* **19**/6, pp. 1059–1063 (1990).

Kaltofen, E., "Polynomial factorization," in *Computer Algebra, 2nd ed.*, edited by B. Buchberger et al.; Springer Verlag, Vienna, pp. 95–113, 1982.

Kaltofen, E., "Polynomial-time reductions from multivariate to bi- and univariate integral polynomial factorization," *SIAM J. Comp.* **14**, pp. 469–489 (1985a).

Kaltofen, E., "Effective Hilbert irreducibility," *Information and Control* **66**, pp. 123–137 (1985b).

Kaltofen, E., "Uniform closure properties of p-computable functions," *Proc. 18th ACM Symp. Theory Comp.*, pp. 330–337 (1986b).

Kaltofen, E., "Greatest common divisors of polynomials given by straight-line programs," *J. ACM* **35**/1, pp. 231–264 (1988).

Kaltofen, E., "Factorization of polynomials given by straight-line programs," in *Randomness and Computation*, Advances in Computing Research **5**, edited by S. Micali; JAI Press, Greenwhich, Connecticut, pp. 375–412, 1989.

Kaltofen, E. and Lakshman, Yagati, "Improved sparse multivariate polynomial interpolation algorithms," *Proc. ISSAC '88, Springer Lect. Notes Comput. Sci.* **358**, pp. 467–474 (1988).

Knuth, D. E., *The Art of Programming, Vol. 2, Semi-Numerical Algorithms, Ed. 2*; Addison Wesley, Reading, MA, 1981.

Landau, S., "Factoring polynomials over algebraic number fields," *SIAM J. Comp.* **14**, pp. 184–195 (1985).

Li, T.-Y., Sauer, T., and Yorke, J. A., "Numerically determining solutions of systems of polynomial equations," *AMS Bulletin* **18**/2, pp. 173–177 (1988).

Macaulay, F. S., "Algebraic theory of modular systems," *Cambridge Tracts* **19**, Cambridge, 1916.

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

Trager, B. M., "Algebraic factoring and rational function integration," *Proc. 1976 ACM Symp. Symbolic Algebraic Comp.*, pp. 219–228 (1976).

Valiant, L., "Reducibility by algebraic projections," *L'Enseignement mathématique* **28**, pp. 253–268 (1982).

van der Waerden, B. L., *Modern Algebra*; F. Ungar Publ. Co., New York, 1953.

Wiedemann, D., "Solving sparse linear equations over finite fields," *IEEE Trans. Inf. Theory* **IT-32**, pp. 54–62 (1986).

Zippel, R. E., "Interpolating polynomials from their values," *J. Symbolic Comput.* **9**/3, pp. 375–403 (1990).

Zulehner, W., "A simple homotopy method for determining all isolated solutions to polynomial systems," *Math. Comp.* **50**/181, pp. 167–177 (1988).

Errata to: Fast Parallel Irreducibility Testing

This Journal (1985) **1**, 57–67

Page 61, line 10: $\lfloor \log_2(K) \rfloor \leftrightarrow |\log_2(K)|$. Page 61, line 24: $\lfloor \log_2 K \rfloor \leftrightarrow |\log_2 K|$.
Page 63, line 8: (6) $\leftrightarrow$ (5). Page 65, line 6: $\lfloor \log_2(n) \rfloor \leftrightarrow |\log_2(n)|$.
Page 65, line 11 from bottom: reducible $\leftrightarrow$ irreducible.
Page 67, Kaltofen (1983) reference: 469–489 $\leftrightarrow$ 469.