On the complexities of multipoint evaluation and interpolation

Alin Bostan and Éric Schost

Laboratoire STIX, École polytechnique, 91128 Palaiseau, France

Abstract

We compare the complexities of multipoint polynomial evaluation and interpolation. We show that, over a field of characteristic zero, both questions have equivalent complexities, up to a constant number of polynomial multiplications.

Key words: Polynomial evaluation, interpolation, complexity.

1 Introduction

Multipoint polynomial evaluation and interpolation are ubiquitous problems. They can be stated as follows:

Evaluation: Given some evaluation points $x_0, \ldots, x_n$ and the coefficients $p_0, \ldots, p_n$ of a polynomial P, compute the values $P(x_i) = \sum_{j=0}^{n} p_j x_i^j$, for $i = 0, \ldots, n$.

Interpolation: Given distinct interpolation points $x_0, \ldots, x_n$ and given some values $q_0, \ldots, q_n$, compute the unique coefficients $p_0, \ldots, p_n$ such that $\sum_{j=0}^{n} p_j x_i^j = q_i$ holds for $i = 0, \ldots, n$.

Note in particular that we are concerned only in dense, univariate evaluation and interpolation algorithms: we shall consider neither multivariate polynomials, nor specific questions arising with sparse polynomials, as for instance in (Ben-Or and Tiwari, 1988).

It is known that the complexities of evaluation and interpolation are closely related: for instance, the interpolation algorithms of Lipson (1971), Fiduccia

Email addresses: `Alin.Bostan@stix.polytechnique.fr` (Alin Bostan), `Eric.Schost@stix.polytechnique.fr` (Éric Schost).

(1972), Moenck and Borodin (1972), Borodin and Moenck (1974) and Strassen (1972/73) all require to perform a multipoint evaluation as a subtask. Thus in this note, rather than describing particular algorithms, we focus on comparing the complexities of both questions, that is, on reductions of one question to the other.

Close links appear when one puts *program transposition* techniques into use. Roughly speaking, such techniques prove that an algorithm that performs a matrix-vector product can be transformed into an algorithm with essentially the same complexity, and which performs the transposed matrix product. These techniques are particularly relevant here, as many relations exist between Vandermonde matrices, their transposes, and other structured matrices such as Hankel matrices.

Using such relations, reductions of interpolation to evaluation, and conversely, have been proposed in, or can be deduced from (Kaltofen and Yagati, 1989; Canny et al., 1989; Pan, 1989; Finck et al., 1993; Gohberg and Olshevsky, 1994a,b; Bini and Pan, 1994; Pan, 2001). Nevertheless, to our knowledge, no equivalence theorem has been established for these questions. All results we are aware of involve the following additional operation: given $x_0, \dots, x_n$, compute the coefficients of $\prod_{i=0}^{n}(T - x_i)$, that is, the elementary symmetric functions in $x_0, \dots, x_n$. If we denote by $\mathsf{E}(n)$, $\mathsf{I}(n)$ and $\mathsf{S}(n)$ the complexities of multipoint evaluation, interpolation and elementary symmetric functions computation on $n + 1$ points, then the above references yield

$$\mathsf{I}(n) \in O(\mathsf{E}(n) + \mathsf{S}(n)) \quad \text{and} \quad \mathsf{E}(n) \in O(\mathsf{I}(n) + \mathsf{S}(n)).$$

The best currently known result gives $\mathsf{S}(n) \in O(\mathsf{M}(n) \log(n))$, where $\mathsf{M}(n)$ is the cost of degree n polynomial multiplication, see (von zur Gathen and Gerhard, 1999, Ch. 10). Thus, the above estimates are of little help, since it is already known that both $\mathsf{E}(n)$ and $\mathsf{I}(n)$ are in $O(\mathsf{M}(n) \log(n))$ (Moenck and Borodin, 1972; Borodin and Moenck, 1974; Strassen, 1972/73; Bostan et al., 2003).

Our purpose in this note is to reduce the gap, replacing the terms $\mathsf{S}(n)$ by $\mathsf{M}(n)$ in the above estimates, in the case when the base field has characteristic zero. With this improvement, such estimates become useful, since for instance they now imply that improving the $O(\mathsf{M}(n) \log(n))$ bound for either evaluation or interpolation entails a similar improvement for the other problem.

Actually, we prove a sharper statement: it is known that evaluation or interpolation simplifies for particular families of points (e.g., geometric progressions), see for instance (Aho et al., 1975; Bostan and Schost, 2003) and the comments below. We take this specificity into account; roughly speaking, we prove that:

- given an algorithm that performs evaluation on some distinguished families of points, one can deduce an algorithm that performs interpolation on the same families of points, and with essentially the same complexity, up to a constant number of polynomial multiplications.

- given an algorithm that performs interpolation on some distinguished families of points, one can deduce an algorithm that performs evaluation on the same families of points, and with essentially the same complexity, up to a constant number of polynomial multiplications.

We can infer two corollaries from these results: first, we deduce the estimates

$$\mathsf{I}(n) \in O(\mathsf{E}(n) + \mathsf{M}(n)) \quad \text{and} \quad \mathsf{E}(n) \in O(\mathsf{I}(n) + \mathsf{M}(n)),$$

as claimed above. Our second corollary relates to results from Aho et al. (1975). That article studies the families of $n + 1$ points in $\mathbb{C}$ on which any degree n polynomial can be evaluated in time $O(\mathsf{M}(n))$. Our results show that these are precisely the families of points on which any degree n polynomial can interpolated in time $O(\mathsf{M}(n))$. For instance, it is proved by Aho et al. (1975) that given any $a, b, c, z \in \mathbb{C}^4$, any degree n polynomial can be evaluated on the sequence $a + bz^i + cz^{2i}$ in time $O(\mathsf{M}(n))$. We deduce that as soon as all these points are distinct, any degree n polynomial can be interpolated on this sequence in time $O(\mathsf{M}(n))$ as well.

Our approach closely follows the ideas given in the references mentioned above, notably (Kaltofen and Yagati, 1989; Canny et al., 1989). We will use reductions of one problem to the other; the underlying ideas are borrowed from these two references. To perform both reductions, we have to compute the symmetric functions in the sample points $x_0, \ldots, x_n$. Technically, we will prove that the cost of this operation reduces to that of either interpolation or evaluation, up to a constant number of polynomial multiplications. To do so, the main ideas are the following:

- Suppose that an algorithm that performs interpolation at $x_0, \ldots, x_n$ is given. We cannot use it to deduce the polynomial $F = \prod_{i=0}^{n}(T - x_i)$ directly, since F has degree $n+1$. Nevertheless, we can recover the polynomial $\prod_{i=1}^{n}(T - x_i)$ by interpolation, since it has degree n, and its values at $x_0, \ldots, x_n$ are easy to compute. Then, recovering F is immediate.
- Suppose that an algorithm that performs evaluation at $x_0, \ldots, x_n$ is given. By transposition, this algorithm can be used to compute the power sums of the polynomial $F = \prod_{i=0}^{n}(T - x_i)$. Then one can deduce the coefficients of F from its power sums using the fast exponentiation algorithm of Brent (1975) and Schönhage (1982).

The rest of this paper is devoted to give a rigorous version of these considerations and their consequences. In the next section, we first precise our computational model, then state our results in this model. Then, we present basic complexity results for polynomials and power series. The next two sections give the proofs of the main theorems and we conclude by discussing a closely related problem.

Finally, let us mention other problems in a similar vein, namely to obtain equivalence results for other evaluation and interpolation questions, notably Newton or Hermite problems. We leave them as further work.

2 Computational model, main result

Our basic computational objects are *straight-line programs* (allowing divisions), which are defined as follows: Let $A = A_0, \dots, A_r$ be a family of indeterminates over a field k. Let us define $g_{-r} = A_0, \dots, g_0 = A_r$. A *straight-line program* Γ is a sequence $g_1, \dots, g_L \subset k(A)$ such that for $1 \leq \ell \leq L$, one of the following holds:

- $g_\ell = \lambda$, with $\lambda \in k$;
- $g_\ell = \lambda \star g_i$, with $\lambda \in k$, $\star \in \{+, -, \times, \div\}$ and $-r \leq i < \ell$;
- $g_\ell = g_i \star g_j$, with $\star \in \{+, -, \times, \div\}$, or $g_\ell = -g_i - g_j$, with in both cases $-r \leq i, j < \ell$.

These rational functions are the *instructions* of Γ. The *size* of Γ is L and it is denoted $s(\Gamma)$; the *output* of Γ is a sequence $G_0, \dots, G_s$ of elements in $\{g_{-r}, \dots, g_L\}$. Γ is *defined* at a point $a = a_0, \dots, a_r \in k^{r+1}$ if a cancels no denominator in $\{g_1, \dots, g_L\}$; in this case, we say that Γ *computes* $(G_i(a))_{0 \leq i \leq s}$ *on input* a.

In the sequel, we have to consider algorithms that take as input both the sample points $x = x_0, \dots, x_n$ and the coefficients (resp. values) of a polynomial P. We will allow arbitrary operations on the sample points. On the other hand, since we compute linear functions of the coefficients (resp. values) of P, we will only allow *linear* operations on them; this is actually not a limitation, because in this case any non-linear step can be simulated by at most 3 linear steps, see (Strassen, 1973) and (Bürgisser et al., 1997, Th. 13.1).

Formally, we will thus consider straight-line programs taking as input two families of indeterminates A and B, allowing only linear operations on the second family of indeterminates. The straight-line programs satisfying these conditions are called *B-linear straight-line programs* (or simply *linear straight-line programs*) and are defined as follows, compare with (Bürgisser et al., 1997, Ch. 13).

Let $A = A_0, \ldots, A_r$ and $B = B_0, \ldots, B_s$ be two families of indeterminates over a field k. Let us define $g_{-r} = A_0, \ldots, g_0 = A_r$ and $\gamma_{-s} = B_0, \ldots, \gamma_0 = B_s$. A *$B$-linear straight-line program* Γ is the data of two sequences $g_1, \ldots, g_L \subset k(A)$ and $\gamma_1, \ldots, \gamma_M \subset k(A)[B]$ such that $g_1, \ldots, g_L$ satisfy the axioms of straight-line programs and for $1 \leq m \leq M$, one of the following holds:

- $\gamma_m = \lambda \gamma_i$, with $\lambda \in k \cup \{g_{-r}, \ldots, g_L\}$ and $-s \leq i < m$;
- $\gamma_m = \pm\gamma_i \pm \gamma_j$, with $-s \leq i, j < m$.

In particular, $\gamma_1, \ldots, \gamma_M$ are linear forms in B, as requested. The sequences $g_1, \ldots, g_L$ and $\gamma_1, \ldots, \gamma_M$ form the *instructions* of Γ. The *size* of Γ is $L + M$, and is denoted $s(\Gamma)$ as above; the *output* of Γ is a sequence $G_0, \ldots, G_s$ of elements of $\{\gamma_{-s}, \ldots, \gamma_M\}$. Γ is *defined* at a point $a = a_0, \ldots, a_r \in k^{r+1}$ if a cancels no denominator in $\{g_1, \ldots, g_L\}$; in this case we say that Γ *computes the linear forms $(G_i(a, B))_{0 \leq i \leq s}$ on input a.*

We use a function denoted by $\mathsf{M}(n)$, which represents the complexity of univariate polynomial multiplication. It is defined as follows: For any $n \geq 0$, let us introduce the indeterminates $A = A_0, \ldots, A_n$, $B = B_0, \ldots, B_n$, and let us define the polynomials $C_0, \ldots, C_{2n}$ in $k[A, B]$ by the relation

$$\left(\sum_{i=0}^{n} A_i T^i\right)\left(\sum_{i=0}^{n} B_i T^i\right) = \sum_{i=0}^{2n} C_i T^i$$

in $k[A, B][T]$. The polynomials C_i are linear in B (they are of course actually bilinear in A, B); then, we require that they can be computed by a *B-linear* straight-line program of size $\mathsf{M}(n)$, that performs no division in the indeterminates A. Again, imposing such conditions is no limitation, since allowing arbitrary operations would at best gain a constant factor. We also suppose that the function M verifies the inequality $\mathsf{M}(n_1)+\mathsf{M}(n_2) \leq \mathsf{M}(n_1+n_2)$ for all $n_1, n_2 \geq 0$. For instance, the algorithms of Schönhage and Strassen (1971) and Cantor and Kaltofen (1991) show that $\mathsf{M}(n)$ can be taken in $O(n \log(n) \log(\log(n)))$.

Main results. With these definitions, our results are the following. Roughly speaking, Theorem 1 shows that, up to a constant number of polynomial multiplications, evaluation is not harder than interpolation, and Theorem 2 proves the converse assertion. As mentioned above, we want to take into account the possibility of specialized algorithms, which may give the result only for some distinguished families of sample points: this is obtained using suitable hypotheses on the points x. All results apply on a field of characteristic zero.

Theorem 1 *Let Γ be a Q-linear straight-line program of size L, taking as input $X = X_0, \ldots, X_n$ and $Q = Q_0, \ldots, Q_n$, and let $G = G_0, \ldots, G_n \in k(X)[Q]$ be the output of Γ. Then there exists a P-linear straight-line program Δ of size*

$2L + O(\mathsf{M}(n))$, taking as input X and $P = P_0, \ldots, P_n$, and with the following property.

Let $x = x_0, \ldots, x_n$ be pairwise distinct points such that Γ is defined at x and such that the sequence $G_j(x, Q)$ satisfies

$$\sum_{j=0}^{n} G_j(x, Q) x_i^j = Q_i, \qquad \text{for } i = 0, \ldots, n.$$

Then Δ is defined at x and the output $H_0, \ldots, H_n$ of Δ satisfies

$$H_i(x, P) = \sum_{j=0}^{n} P_j x_i^j, \qquad \text{for } i = 0, \ldots, n.$$

Theorem 2 *Let Δ be a P-linear straight-line program of size L, taking as input $X = X_0, \ldots, X_n$ and $P = P_0, \ldots, P_n$, and let $H_0, \ldots, H_n \in k(X)[P]$ be the output of Δ. Then there exists a Q-linear straight-line program Γ of size $3L + O(\mathsf{M}(n))$, taking as input X and $Q = Q_0, \ldots, Q_n$, and with the following property.*

Let $x = x_0, \ldots, x_n$ be pairwise distinct points such that Δ is defined at x and such that the sequence $H_i(x, P)$ satisfies

$$H_i(x, P) = \sum_{j=0}^{n} P_j x_i^j, \qquad \text{for } i = 0, \ldots, n.$$

Then Γ is defined at x and the output $G_0, \ldots, G_n$ of Γ satisfies

$$\sum_{j=0}^{n} G_j(x, Q) x_i^j = Q_i, \qquad \text{for } i = 0, \ldots, n.$$

3 Preliminaries

In this section, we present preliminary results that are needed for what follows. The first of them is our basic tool, that relates the complexity of computing a linear map to that of its transpose. Next, we recall some basic complexity results for power series and polynomials. Finally, we describe how complexity behaves through the composition or evaluation of rational functions or linear forms.

All straight-line programs considered below are defined over some field k; we suppose that k has characteristic zero so as to be able to apply some fast algorithms of Brent (1975) and Schönhage (1982).

3.1 Program transposition

Inspired by (Kaltofen and Yagati, 1989; Canny et al., 1989; Pan, 2001), we will use the following idea: any algorithm that performs interpolation (resp. evaluation) can be transformed into one that performs the transposed operation. Originating from (Bordewijk, 1956), and sometimes referred to as Tellegen's theorem (Tellegen, 1952), the *transposition principle* precisely gives this kind of result, and predicts the difference of complexity induced by the transposition operation; see (Bürgisser et al., 1997) for a proof and (Kaltofen, 2000) for a detailed discussion. In our context, we easily obtain the following result:

Lemma 1 *Let Γ be P-linear straight line program of size L, taking as input $X = X_0, \ldots, X_n$ and $P = P_0, \ldots, P_n$ and let $G = G_0, \ldots, G_n \in k(X)[P]$ be the output of Γ. Then there exists a Q-linear straight line program $\Gamma^\dagger$ of size $L + O(n)$, with input X and $Q = Q_0, \ldots, Q_n$, with output $H = H_0, \ldots, H_n$, and with the following property.*

Let $x \in k^{n+1}$ be such that Γ is defined at x and let $\varphi : k^{n+1} \to k^{n+1}$ be the linear map $p \mapsto G(x, p)$. Then $\Gamma^\dagger$ is defined at x and $q \mapsto H(x, q)$ is the transposed map of φ.

3.2 Polynomial and power series algorithms

In what follows, we need to perform basic operations on polynomials, such as recovering a polynomial from its Newton sums and conversely. We now discuss fast algorithms for such questions, of complexity bounded by a constant times that of polynomial multiplication.

Let first F be a polynomial of degree $n+1$ in $k[T]$. Writing $F = \prod_{i=0}^{n}(T - x_i)$ over an algebraic closure of k, the ith *Newton sum* of F is defined as $\sum_{j=0}^{n} x_j^i$ (so the 0th Newton sum is $n + 1$). Our question will be to compute the first $2n + 1$ Newton sums of F. The following lemma gives a complexity estimate for this operation, using the fact that the generating series at infinity of the Newton sums of F is the logarithmic derivative of F, see (Schönhage, 1982).

Lemma 2 *Let $n \in \mathbb{N}$. There exists a straight-line program P_n with input $F_0, \ldots, F_n$, with output $A_0, \ldots, A_{2n}$ and with the following property. For all $f = f_0, \ldots, f_n \in k^{n+1}$, P_n is defined at f and for $0 \le i \le 2n$, $A_i(f)$ is the ith Newton sum of the polynomial $\sum_{i=0}^{n} f_i T^i + T^{n+1}$. Furthermore, the size of P_n is in $O(\mathsf{M}(n))$.*

Conversely, we ask the question of recovering a monic polynomial of degree $n+1$ from its first Newton sums. In characteristic zero, Newton formulas allow one to do this, but using them has a complexity quadratic in n. The following lemma shows that better can be done; this result originates from (Schönhage, 1982) and uses the exponentiation algorithm of Brent (1975).

Lemma 3 *Let $n \in \mathbb{N}$. There exists a straight-line program N_n with input $A_1, \ldots, A_{n+1}$, with output $F_0, \ldots, F_n$ and with the following property. For all $a = a_1, \ldots, a_{n+1} \in k^{n+1}$, N_n is defined at a and for $1 \le i \le n+1$, a_i is the ith Newton sum of the polynomial $\sum_{i=0}^{n} F_i(a)T^i + T^{n+1}$. Furthermore, the size of N_n is in $O(\mathsf{M}(n))$.*

Next, the following lemma states that a matrix-vector product by a Hankel matrix can be performed in time proportional to that of polynomial multiplication. This result is classical, see for instance (Bini and Pan, 1994).

Lemma 4 *Let $n \in \mathbb{N}$. There exists a A-linear straight-line program H_n with input $S_0, \ldots, S_{2n}$ and $A_0, \ldots, A_n$, with output $H_0, \ldots, H_n$ and with the following property. The size of H_n is in $O(\mathsf{M}(n))$; for all $s = s_0, \ldots, s_{2n}$ in $k^{2n+1}$and $a = a_0, \ldots, a_n$ in k^{n+1}, H_n is defined at a and we have*

$$\begin{bmatrix} s_0 & \ldots & s_n \\ \vdots & & \vdots \\ s_n & \ldots & s_{2n} \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} H_0(s,a) \\ \vdots \\ H_n(s,a) \end{bmatrix}.$$

Let finally $n \in \mathbb{N}$, and $A = A_0, \ldots, A_n$ and $B = B_0, \ldots, B_n$ be indeterminates. Then we will denote by $\mathsf{MulTrunc}_n$ a B-linear straight-line program that outputs the coefficients of $\left(\sum_{i=0}^{n} A_i T^i\right)\left(\sum_{i=0}^{n} B_i T^i\right)$ modulo T^{n+1}, has size $\mathsf{M}(n)$, and performs no division in the indeterminates A.

3.3 Composition rules

In the following sections, we will design algorithms from basic building blocks, such as polynomial multiplication, or the algorithms mentioned above. Seeing the output of an algorithm as a sequence of rational functions or linear forms, such constructions correspond to composition. We now define the corresponding rules for (linear) straight-line programs.

Let $X = X_0, \ldots, X_n$ and $P = P_0, \ldots, P_m$ be two sets of indeterminates. There are several ways to compose or evaluate rational functions in $k(X)$ and linear forms in $k(X)[P]$. We now review them, and show how to translate these operations at the level of (linear) straight-line programs. Though technical, these definitions bear no difficulty. We leave it to the reader to check that in all cases, the axioms of (linear) straight-line programs are satisfied.

- We first consider the composition of rational functions. Let then $G = G_0, \ldots, G_n$ and $G' = G'_0, \ldots, G'_s$ be in $k(X)$, and let us write $G'(G) = (G'_i(G_0, \ldots, G_n))_{0 \le i \le s}$. Let also Γ and Γ' be straight-line programs whose outputs are G and G'; we now define a straight-line program that computes $G'(G)$.

  Let $g_1, \ldots, g_L$ be the instructions of Γ and $g'_1, \ldots, g'_{L'}$ those of Γ'. For $1 \le i \le L'$, let $g_{i+L} = g'_i(G_0, \ldots, G_n) \in k(X)$. We let $\Gamma' \circ \Gamma$ be the straight-line program with instructions $g_1, \ldots, g_{L+L'}$ and output $G'(G)$. Then, $s(\Gamma' \circ \Gamma) = s(\Gamma') + s(\Gamma)$.

- We can also compose linear forms. Let then $G = G_0, \ldots, G_m$, let $G' = G'_0, \ldots, G'_s$ be linear forms in $k(X)[P]$, and write $G'(G) = (G'_i(G_0, \ldots, G_m))_{0 \le i \le s}$. Let Γ and Γ' be P-linear straight-line programs whose outputs are G and G'; we now define a P-linear straight-line program that computes $G'(G)$.

  Let $g_1, \ldots, g_L$ and $\gamma_1, \ldots, \gamma_M$ be the instructions of Γ and $g'_1, \ldots, g'_{L'}$ and $\gamma'_1, \ldots, \gamma'_{M'}$ those of Γ'. For $1 \le i \le L'$, let $g_{i+L} = g'_i$; for $1 \le i \le M'$, let γ_{i+M} be the linear form $\gamma'_i(G_0, \ldots, G_m)$ obtained by composition. We let $\Gamma' \bullet \Gamma$ be the P-linear straight-line program with instructions $g_1, \ldots, g_{L+L'}$ and $\gamma_1, \ldots, \gamma_{M+M'}$, and output $G'(G)$. Then, $s(\Gamma' \bullet \Gamma) = s(\Gamma') + s(\Gamma)$.

- We next evaluate linear forms on rational functions. Let $G = G_0, \ldots, G_m$ in $k(X)$, let $G' = G'_0, \ldots, G'_s$ be linear forms in $k(X)[P]$, and write $G'(G) = (G'_i(G_0, \ldots, G_m))_{0 \le i \le s}$, which are in $k(X)$. Let also Γ be a straight-line program and Γ' a P-linear straight-line program, whose outputs are G and G'; we now define a straight-line program that computes $G'(G)$.

  Let $g_1, \ldots, g_L$ be the instructions of Γ and $g'_1, \ldots, g'_{L'}$ and $\gamma'_1, \ldots, \gamma'_{M'}$ those of Γ'. For $1 \le i \le L'$, let $g_{i+L} = g'_i$; for $1 \le i \le M'$ let $g_{i+L+L'}$ be the rational function $\gamma'_i(G_0, \ldots, G_m) \in k(X)$ obtained by evaluation. We let $\Gamma' \star \Gamma$ be the straight-line program with instructions $g_1, \ldots, g_{L+L'+M'}$ and output $G'(G)$. Then, $s(\Gamma' \star \Gamma) = s(\Gamma') + s(\Gamma)$.

- Let finally $G = G_0, \ldots, G_n$ be in $k(X)$ and $G' = G'_0, \ldots, G'_s$ be linear forms in $k(X)[P]$. For any linear form $g \in k(X)[P]$, writing $g = \sum_{0 \le i \le m} g_i P_i$ with all $g_i \in k(X)$, we define $g(G, P) = \sum_{0 \le i \le m} g_i(G_0, \ldots, G_n) P_i$. Then, we define $G'(G, P) = (G'_i(G, P))_{0 \le i \le s}$.

  Let Γ be a straight-line program and Γ' a P-linear straight-line program, whose outputs are G and G'; we now define a P-linear straight-line program that computes $G'(G, P)$.

  Let $g_1, \ldots, g_L$ be instructions of Γ and $g'_1, \ldots, g'_{L'}$ and $\gamma'_1, \ldots, \gamma'_{M'}$ those of Γ'. For $1 \le i \le L'$, let $g_{i+L} = g_i(G_0, \ldots, G_n)$; for $1 \le i \le M'$, let $\overline{\gamma}_i = \gamma'_i(G, P)$. We let $\Gamma' \diamond \Gamma$ be the P-linear straight-line programs with instructions $g_1, \ldots, g_{L+L'}$ and $\overline{\gamma}_1, \ldots, \overline{\gamma}_{M'}$, and output $G'(G, P)$. Then, $s(\Gamma' \diamond \Gamma) = s(\Gamma') + s(\Gamma)$.

4 From interpolation to evaluation

We now prove Theorem 1: given an algorithm that performs interpolation, possibly on some distinguished families of points only, one deduces an algorithm which performs evaluation, on the same families of points, and with essentially the same complexity. The reduction is based on the following matrix identity, which appeared in (Canny et al., 1989):

$$\begin{bmatrix} 1 & \dots & 1 \\ \vdots & & \vdots \\ x_0^n & \dots & x_n^n \end{bmatrix} \begin{bmatrix} 1 & \dots & x_0^n \\ \vdots & & \vdots \\ 1 & \dots & x_n^n \end{bmatrix} = \begin{bmatrix} s_0 & \dots & s_n \\ \vdots & & \vdots \\ s_n & \dots & s_{2n} \end{bmatrix},$$

where $s_i = \sum_{j=0}^n x_j^i$ is the ith Newton sum of $F = \prod_{i=0}^n (T - x_i)$. We rewrite this identity as $(V^t)V = H$, where H is the Hankel matrix made upon $s_0, \dots, s_{2n}$ and V the Vandermonde matrix made upon $x_0, \dots, x_n$. This in turn yields $V = (V^t)^{-1}H$.

Using this last equality, we deduce the following algorithm to evaluate a polynomial P on the points $x_0, \dots, x_n$; this algorithm appeared originally in (Canny et al., 1989) in a "transposed" form, see also (Pan, 1989).

(1) Compute the Newton sums $s_0, \dots, s_{2n}$ of $F = \prod_{i=0}^n (T - x_i)$.

(2) Compute $p' = Hp$, where H is defined as above and p is the vector of coefficients of P.

(3) Compute $(V^t)^{-1}p'$.

Our contribution is the remark that the first step can be essentially reduced to perform a suitable interpolation. Consider indeed $f = \prod_{i=1}^n (T - x_i)$. Then we have the equalities

$$f(x_0) = \prod_{i=1}^n (x_0 - x_i) \quad \text{and} \quad f(x_i) = 0, \ i > 0.$$

The value $f(x_0)$ can be computed in $O(n)$ operations. It then suffices to interpolate the values $f(x_i)$ at $x_0, \dots, x_n$ to recover the coefficients of f, since this polynomial has degree n. Then, the coefficients of $F = (T - x_0)f$ can be deduced for $O(n)$ additional operations. Finally, we can compute the first $2n+1$ Newton sums of F for $O(\mathsf{M}(n))$ additional operations following Lemma 2; this concludes the description of Step 1.

On input the Newton sums $s_0, \dots, s_{2n}$ and the coefficients of P, Step 2 can be done in time $O(\mathsf{M}(n))$ since H is a Hankel matrix. It then suffices to perform a transposed interpolation to conclude Step 3. To summarize, our algorithm requires one interpolation and one transposed interpolation at $x_0, \ldots, x_n$, and $O(\mathsf{M}(n))$ additional operations; in view of Lemma 1, this gives Theorem 1.

Let us now give a formal proof of our assertions. Let Γ be a linear straight-line program of size L as in Theorem 1, and $\Gamma^\dagger$ the linear straight-line program obtained by applying Lemma 1 to Γ. Let x be as in Theorem 1 and let P_n and H_n be as in Section 3.2.

Let next η_1 be a straight-line program performing $O(n)$ additions and multiplications, with input $X_0, \ldots, X_n$ and output $\prod_{i=1}^n (X_0 - X_i), 0, \ldots, 0$ and let $\eta_2 = \Gamma \star \eta_1$. Then on input x, η_2 computes the coefficients of the polynomial f defined above.

Let η_3 be obtained by adding $O(n)$ additions and multiplications to η_2, so as to compute the coefficients of F, and let $\eta_4 = \mathsf{P}_n \circ \eta_3$. Then on input x, η_4 computes the first $2n+1$ Newton sums of F.

We finally define $\eta_5 = \mathsf{H}_n \diamond \eta_4$ and $\eta_6 = \Gamma^\dagger \bullet \eta_5$. Then on input x, η_5 computes the linear forms $p'_0, \ldots, p'_n$ defined above, and η_6 computes the values of P at the points p. The size estimates given in Section 3 show that the size of η_6 is $2L + O(\mathsf{M}(n))$, as requested, concluding the proof.

Finally, we note that the idea of using interpolation algorithms to compute the elementary symmetric functions (in the context of bounded-depth arithmetic circuits) is attributed to Ben-Or by Grolmusz (2003).

5 From evaluation to interpolation

We finally prove Theorem 2: given an algorithm that performs evaluation, possibly on some distinguished families of points only, one deduces an algorithm which performs interpolation, on the same families of points, and with essentially the same complexity. Consider the matrix-vector product

$$\begin{bmatrix} 1 & \ldots & x_0^n \\ \vdots & & \vdots \\ 1 & \ldots & x_n^n \end{bmatrix} \begin{bmatrix} p_0 \\ \vdots \\ p_n \end{bmatrix} = \begin{bmatrix} q_0 \\ \vdots \\ q_n \end{bmatrix}.$$

Our goal is to compute $p = p_0, \ldots, p_n$ on input q. To do so, we first consider the transposed problem, that is, computing $p' = p'_0, \ldots, p'_n$ on input q, where

p' is given by

$$\begin{bmatrix} 1 & \dots & 1 \\ \vdots & & \vdots \\ x_0^n & \dots & x_n^n \end{bmatrix} \begin{bmatrix} p'_0 \\ \vdots \\ p'_n \end{bmatrix} = \begin{bmatrix} q_0 \\ \vdots \\ q_n \end{bmatrix}. \tag{1}$$

To solve this question, we use a reduction that appeared in (Kaltofen and Yagati, 1989) (see also (Pan, 2001) for an alternative formula originating from (Gohberg and Olshevsky, 1994a), which requires essentially the same operations). It is easily checked that the generating series $\mathcal{Q} = \sum_{i=0}^{n} q_i T^i$ satisfies the following identity:

$$\mathcal{Q} \cdot \prod_{i=0}^{n} (1 - x_i T) = \sum_{i=0}^{n} \left(p'_i \prod_{0 \le j \le n}^{j \ne i} (1 - x_j T) \right) \mod T^{n+1}.$$

Define

$$F = \prod_{i=0}^{n} (T - x_i) \quad \text{and} \quad G = T^{n+1} F(1/T) = \prod_{i=0}^{n} (1 - x_i T),$$

$$H = \sum_{i=0}^{n} \left(p'_i \prod_{0 \le j \le n}^{j \ne i} (1 - x_j T) \right) \quad \text{and} \quad I = T^n H(1/T) = \sum_{i=0}^{n} \left(p'_i \prod_{0 \le j \le n}^{j \ne i} (T - x_j) \right).$$

Then we have $H = \mathcal{Q}G \mod T^{n+1}$ and $p'_i = I(x_i)/F'(x_i)$. We deduce the following algorithm for recovering $p'_0, \dots, p'_n$ from $q_0, \dots, q_n$. This originally appeared in (Kaltofen and Yagati, 1989) and follows (Zippel, 1990).

(1) Compute $F = \prod_{i=0}^{n} (T - x_i)$ and $G = T^{n+1} F(1/T)$.

(2) Compute $H = \mathcal{Q}G \mod T^{n+1}$ and $I = T^n H(1/T)$.

(3) Evaluate I and F' on $x_0, \dots, x_n$ and output $I(x_i)/F'(x_i)$.

As in the previous section, our contribution concerns Step 1: we show that computing F is not more costly than performing an evaluation and some polynomial multiplications.

Indeed, let us compute the transposed evaluation on the set of points $x_0, \dots, x_n$ with input values $x_0, \dots, x_n$: this gives the first Newton sums of F, $\sum_{i=0}^{n} x_i^j$, for $1 \le j \le n+1$. Then following Lemma 3 we can recover the coefficients of the polynomial F for $O(\mathsf{M}(n))$ operations. This concludes the description of Step 1.

Step 2 can then be done for $\mathsf{M}(n)$ operations, and Step 3 for two multipoint evaluations plus $n+1$ scalar divisions. This algorithm thus requires two evaluations and one transposed evaluation at $x_0, \ldots, x_n$, and $O(\mathsf{M}(n))$ additional operations. Transposing backwards answers our question.

We now give a formal proof of Theorem 2. Let Δ be a linear straight-line program of size L as in Theorem 2 and $\Delta^\dagger$ be obtained by applying Lemma 1 to Δ. We next take x as in Theorem 2. Let finally N_n be as in Section 3.2 and X the straight-line program of size 0 that has $X_0, \ldots, X_n$ for input and output.

We first define $\delta_1 = \Delta^\dagger \star \mathsf{X}$ and $\delta_2 = \mathsf{N}_n \circ \delta_1$. Then on input x, δ_2 computes the coefficients of F. By adding $O(n)$ operations to δ_2, we define a straight-line program δ_3 that computes the coefficients of F'; by reversing the order of the output of δ_2, we define a straight-line program δ_4 that computes the coefficients of G.

Let now $\mathsf{MulTrunc}_n$ be as in Section 3.2 and define $\delta_5 = \mathsf{MulTrunc}_n \diamond \delta_4$; then on input x, δ_5 computes the coefficients of H. By reversing the order of the output of δ_5, we define a linear straight-line program δ_6 that computes the coefficients of I.

Next, let us introduce $\delta_7 = \Delta \bullet \delta_6$ and $\delta_8 = \Delta \star \delta_3$. On input x, they respectively compute the values $I(x_i)$ and $F'(x_i)$. Let finally Div be the linear straight-line program that takes X, Q as input and outputs $Q_0/X_0, \ldots, Q_n/X_n$. We conclude by defining $\delta_9 = \mathsf{Div} \diamond \delta_8$ and $\delta_{10} = \delta_9 \bullet \delta_7$. Then on input x, δ_{10} computes the values p' defined above. By the results of Section 3, it has size $3L + O(\mathsf{M}(n))$. Applying Lemma 1 to δ_{10} concludes the proof.

6 Further results

Given $x = (x_0, \ldots, x_n)$, let us denote by $\mathsf{LinComb}_x$ the following operation of linear combination:

$$c = (c_0, \ldots, c_n) \in k^{n+1} \longmapsto \sum_{i=0}^{n} c_i \prod_{0 \le j \le n}^{j \ne i} (T - x_j)$$

The complexities of this operation and those of multipoint evaluation and interpolation are closely related: the classical interpolation algorithms use this operation as a subtask (which was also used in the previous section). To conclude this paper, we will establish that this operation has a complexity equivalent to evaluation and interpolation, up to suitable correcting terms in $O(\mathsf{M}(n))$. We will keep our discussion informal, leaving it to the reader to formalize these arguments in our complexity model. In what follows, we write $F = \prod_{i=0}^{n}(T - x_i)$.

From linear combination to multipoint evaluation. Suppose that an algorithm that performs the $\mathsf{LinComb}$ operation at $x = x_0, \ldots, x_n$ is given. We show how to deduce an algorithm for evaluation at x.

Applying $\mathsf{LinComb}_x$ to the vector $(1, 0, \ldots, 0)$, we obtain the coefficients of the polynomial $f = F/(x - x_0)$; then, the polynomial F can be recovered from f using $O(n)$ additional operations. Suppose now that we want to evaluate a polynomial P at x. Let $G = T^{n+1}F(1/T)$, $Q = T^n P(1/T)$ and $R = Q/G$ modulo T^{n+1}. Then it was shown by Bostan et al. (2003) that the values $P(x_0), \ldots, P(x_n)$ are obtained by applying the transpose of $\mathsf{LinComb}_x$ to the polynomial R. A power series division at precision $n+1$ requires $O(\mathsf{M}(n))$ operations. Using Lemma 1, we deduce that the complexity of multipoint evaluation at x is bounded from above by twice the complexity of performing $\mathsf{LinComb}$ at x and $O(\mathsf{M}(n))$ additional operations.

From interpolation to linear combination. Suppose that an algorithm that performs interpolation at $x = x_0, \ldots, x_n$ is given. We show how to deduce an algorithm for performing the linear combination at x.

The Lagrange interpolation formula implies that the matrix of the linear combination equals

$$\begin{bmatrix} 1 & \ldots & x_0^n \\ \vdots & & \vdots \\ 1 & \ldots & x_n^n \end{bmatrix}^{-1} \begin{bmatrix} F'(x_0) & \ldots & 0 \\ & \ddots & \\ 0 & \ldots & F'(x_n) \end{bmatrix}.$$

On the other hand, the values $F'(x_0), \ldots, F'(x_n)$ can be recovered by performing a transposed interpolation at x, due to the equality

$$\begin{bmatrix} 1 & \ldots & 1 \\ \vdots & & \vdots \\ x_0^n & \ldots & x_n^n \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ \vdots \\ 1 \end{bmatrix} = \begin{bmatrix} 1/F'(x_0) \\ \vdots \\ 1/F'(x_n) \end{bmatrix}.$$

These matrix equalities show that performing the linear combination amounts to a transposed interpolation, followed by a direct interpolation at x. Thus, the complexity of $\mathsf{LinComb}_x$ is bounded from above by twice that of interpolation at x and $O(n)$ additional operations.

7 Acknowledgments

We wish to thank Richard Brent, Bruno Salvy and Gilles Villard and one anonymous referee for their careful reading of this paper, and Jürgen Gerhard, whose question on the possible existence of an "inversion principle" stimulated this research.

References

Aho, A. V., Steiglitz, K., Ullman, J. D., 1975. Evaluating polynomials at fixed sets of points. SIAM Journal on Computing 4 (4), 533–539.

Ben-Or, M., Tiwari, P., 1988. A deterministic algorithm for sparse multivariate polynomial interpolation. In: STOC'88. ACM Press, pp. 301–309.

Bini, D., Pan, V. Y., 1994. Polynomial and matrix computations. Vol. 1. Progress in Theoretical Computer Science. Birkhäuser Boston Inc.

Bordewijk, J. L., 1956. Inter-reciprocity applied to electrical networks. Appl. Sci. Res. B. 6, 1–74.

Borodin, A., Moenck, R., 1974. Fast modular transforms. Journal of Computer and System Sciences 8, 366–386.

Bostan, A., Lecerf, G., Schost, É., 2003. Tellegen's principle into practice. In: ISSAC'03. ACM Press, pp. 37–44.

Bostan, A., Schost, É., 2003. Evaluation and interpolation on special sets of points. Tech. rep., École polytechnique.

Brent, R. P., 1975. Multiple-precision zero-finding methods and the complexity of elementary function evaluation. Analytic Computational Complexity, 151–176.

Bürgisser, P., Clausen, M., Shokrollahi, M. A., 1997. Algebraic complexity theory. Vol. 315 of Grundlehren der Mathematischen Wissenschaften. Springer–Verlag.

Canny, J., Kaltofen, E., Yagati, L., 1989. Solving systems of non-linear polynomial equations faster. In: ISSAC'89. ACM Press, pp. 121–128.

Cantor, D. G., Kaltofen, E., 1991. On fast multiplication of polynomials over arbitrary algebras. Acta Informatica 28 (7), 693–701.

Fiduccia, C. M., 1972. Polynomial evaluation via the division algorithm: The fast Fourier transform revisited. In: Conference Record, Fourth Annual ACM Symposium on Theory of Computing. pp. 88–93.

Finck, T., Heinig, G., Rost, K., 1993. An inversion formula and fast algorithms for Cauchy-Vandermonde matrices. Linear Algebra and its Applications 183, 179–191.

von zur Gathen, J., Gerhard, J., 1999. Modern computer algebra, 1st Edition. Cambridge University Press.

Gohberg, I., Olshevsky, V., 1994a. Complexity of multiplication with vectors for structured matrices. Linear Algebra and its Applications 202, 163–192.

Gohberg, I., Olshevsky, V., 1994b. Fast algorithms with preprocessing for matrix-vector multiplication problems. Journal of Complexity 10 (4), 411–427.

Grolmusz, V., 2003. Computing elementary symmetric polynomials with a subpolynomial number of multiplications. SIAM Journal on Computing 32 (6), 1475–1487.

Kaltofen, E., 2000. Challenges of symbolic computation: my favorite open problems. With an additional open problem by Robert M. Corless and David J. Jeffrey. Journal of Symbolic Computation 29 (6), 891–919.

Kaltofen, E., Yagati, L., 1989. Improved sparse multivariate polynomial interpolation algorithms. In: ISSAC'88. Vol. 358 of Lecture Notes in Computer Science. Springer Verlag, pp. 467–474.

Lipson, J. D., 1971. Chinese remainder algorithm and interpolation algorithms. In: Proceedings 2nd ACM Symposium of Symbolic and Algebraic Manipulation. ACM Press, pp. 372–391.

Moenck, R. T., Borodin, A., 1972. Fast modular transforms via division. Thirteenth Annual IEEE Symposium on Switching and Automata Theory (Univ. Maryland, College Park, Md., 1972), 90–96.

Pan, V. Y., 1989. On computations with dense structured matrices. In: ISSAC'89. ACM Press, pp. 34–42.

Pan, V. Y., 2001. Structured matrices and polynomials. Birkhäuser Boston Inc., Boston, MA.

Schönhage, A., 1982. The fundamental theorem of algebra in terms of computational complexity. Tech. rep., University of Tübingen.

Schönhage, A., Strassen, V., 1971. Schnelle Multiplikation großer Zahlen. Computing 7, 281–292.

Strassen, V., 1972/73. Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten. Numerische Mathematik 20, 238–251.

Strassen, V., 1973. Vermeidung von Divisionen. Journal für die reine und angewandte Mathematik 264, 184–202.

Tellegen, B., 1952. A general network theorem, with applications. Technical Report 7, 259–269, Philips Research.

Zippel, R., 1990. Interpolating polynomials from their values. Journal of Symbolic Computation 9 (3), 375–403.