Factorization of Polynomials and GCD Computations for Finding Universal Denominators*

S.A. Abramov[1], A. Gheffar[2], and D.E. Khmelnov[1]

[1] Computing Centre of the Russian Academy of Sciences, Vavilova, 40, Moscow 119991, GSP-1 Russia
sergeyabramov@mail.ru, dennis_khmelnov@mail.ru
[2] Institute XLIM, Université de Limoges, CNRS, 123, Av. A. Thomas, 87060 Limoges cedex, France
f_gheffar@yahoo.fr

Abstract. We discuss the algorithms which, given a linear difference equation with rational function coefficients over a field k of characteristic 0, compute a polynomial $U(x) \in k[x]$ (a universal denominator) such that the denominator of each of rational solutions (if exist) of the given equation divides $U(x)$. We consider two types of such algorithms. One of them is based on constructing a set of irreducible polynomials that are candidates for divisors of denominators of rational solutions, and on finding a bound for the exponent of each of these candidates (the full factorization of polynomials is used). The second one is related to earlier algorithms for finding universal denominators, where the computation of gcd was used instead of the full factorization. The algorithms are applicable to scalar equations of arbitrary orders as well as to systems of first-order equations.

A complexity analysis and a time comparison of the algorithms implemented in Maple are presented.

1 Introduction

In the early 1990s, computer algebra researchers and programmers tried not to use the complete (full) factorization of polynomials unless it was inevitable since this operation was very costly. Designing an algorithm everybody tried to find a suitable type of incomplete factorization based on computation of the greatest common divisors (gcd's) following classical samples of M.V.Ostrogradsky's and Ch.Hermite's algorithms for extracting the rational part of an indefinite integral of rational function. But later the situation with full factorization algorithms changed. Currently very fast and practical algorithms have become known, — see, e.g., [16]. Of course the complexity of the algorithms for the full factorization grows faster than the complexity of the algorithms for computing gcd when polynomial degrees tend to infinity. But when the degrees are of moderate size

* Supported by ECONET grant 21315ZF.

© Springer-Verlag Berlin Heidelberg 2010

the full factorization is not costlier than the computation of gcd, e.g., in Maple system [22]. Thus, an interesting general problem arises, namely the problem of designing new alternative computer algebra algorithms based on the full factorization instead of numerous calls for the gcd subroutine. The appropriateness of such alternative algorithms has to be carefully investigated for any particular relevant computer algebra problem. Such investigation must be supported by suitable correct experiments.

In this paper, we revisit a problem related to the search for rational solutions of a linear difference equation with polynomial coefficients. Rational solutions may be a building block for other types of solutions, and more general, such algorithms may be a part of various computer algebra algorithms (see [21], [8], [9], [17], etc.). As a consequence, investigations of new ways to construct such solutions are quite valuable for computer algebra.

Let k be a field of characteristic 0. We consider systems of the form

$$Y(x+1) = A(x)Y(x), \tag{1}$$

$Y(x) = (Y_1(x), Y_2(x), \dots, Y_n(x))^T$, $A(x) = (a_{ij}(x)) \in \mathrm{Mat}_n(k(x))$. It is assumed that there exists the inverse matrix $A^{-1}(x) = (\tilde{a}_{ij}(x)) \in \mathrm{Mat}_n(k(x))$. If an inhomogeneous system $Y(x+1) = A(x)Y(x) + G(x)$ is given and $A(x)$ is as in (1), $G(x) \in k(x)^n$, then by adding to $Y(x)$ an $(n+1)$-st component with value 1, one can transform the given system into a homogeneous system with an invertible matrix $B(x) \in \mathrm{Mat}_{n+1}(k(x))$ (see, e.g., [15, Sect. 2.2]). For this reason we restrict our consideration to (1). At the same time we will consider scalar equations of the form

$$y(x+n) + a_{n-1}(x)y(x+n-1) + \ldots + a_1(x)y(x+1) + a_0(x)y(x) = \varphi(x), \tag{2}$$

$\varphi(x), a_1(x), \dots, a_{n-1}(x) \in k(x)$, $a_0(x) \in k(x) \setminus \{0\}$, and such an equation is inhomogeneous if $\varphi(x)$ is a non-zero rational function. By clearing denominators we can rewrite (2) as

$$b_n(x)y(x+n) + \ldots + b_1(x)y(x+1) + b_0(x)y(x) = \psi(x), \tag{3}$$

$\psi(x), b_1(x), \dots, b_{n-1}(x) \in k[x]$, $b_0(x), b_n(x) \in k[x] \setminus \{0\}$.

Currently, a few algorithms for finding rational (i.e., rational function) solutions of equations (2), (3) and systems (1) are known. The algorithms from [5,6,11,14] first construct a *universal denominator*, i.e., a polynomial $U(x)$ such that in the scalar case an arbitrary rational solution $y(x)$ of (2) or (3) can be represented as

$$y(x) = \frac{z(x)}{U(x)}, \tag{4}$$

where $z(x) \in k[x]$ (in other words, if (2) has a rational solution $\frac{f(x)}{g(x)}$ which is in the lowest terms then $g(x) | U(x)$). In the case of a system an arbitrary rational solution of (1) can be represented as

$$Y_i(x) = \frac{Z_i(x)}{U(x)}, \quad i = 1, 2, \dots, n, \tag{5}$$

where $Z_1(x), Z_2(x), \dots, Z_n(x) \in k[x]$.

The algorithm from [14] is based on constructing a set of irreducible polynomials that are candidates for divisors of denominators of rational solutions, and on finding in a quite simple way a bound for the exponent of each of these candidates. Such algorithms use the full factorization of polynomials. Experiments with the Maple system show that the full factorization makes some of computer algebra algorithms significantly faster in comparison with algorithms based on computations of gcd's and resultants ([20], [10] etc.).

When a universal denominator is constructed, one can substitute (4), (5) with undetermined $z(x)$ resp. $Z_i(x)$ into the initial equation resp. system to reduce the problem of searching for rational solutions to the problem of searching for polynomial solutions. After this, e.g., the algorithms from [2,7] (the scalar case; see also [13, Sect. 9]) and the corresponding algorithm from [6,11,18] (the case of a system) can be used.

The algorithm from [15] is applicable to the system (1) when $k = \mathbb{C}$. It finds n rational functions $R_1(x), R_2(x), \dots, R_n(x) \in \mathbb{C}(x)$ which are called *bounds for denominators* such that for any rational solution of (1) we have

$$Y_i(x) = Z_i(x)R_i(x), \quad i = 1, 2, \dots, n, \tag{6}$$

where $Z_1(x), Z_2(x), \dots, Z_n(x) \in \mathbb{C}[x]$ (the numerator of $R_i(x)$ is a factor of the numerator of the ith entry $Y_i(x)$ of any rational solution $Y(x)$). The substitution (6) is used instead of (4), (5). The algorithm from [15] can lead to a more "productive" substitution. But the general situation is not so simple. This algorithm is based on matrix operations (matrix entries are in $\mathbb{C}(x)$) which are costly. It is shown in [14, Th. 2] that there exist such examples when substitutions (5), (6) are identical, but the algorithm from [15], spends much more time than the algorithms from [5,6,11,14].

In this paper we concentrate on the approach discussed in [14].

The paper is organized as follows. Section 2 is devoted to a theoretical basis for algorithms for constructing universal denominators (a short review). Section 3 contains descriptions of the algorithm from [5,6,11,14]. In addition, we propose an improved version of the algorithm from [14]. In Section 4 we give some analysis of these algorithms and prove that all of them give the same universal denominator. A complexity analysis is given as well. In Section 5, we discuss our implementation of the proposed improved version of the algorithm from [14]. Section 6 contains a time comparison of this algorithm with the algorithms from [5,6] which are exploited in current versions of Maple. Finally in Section 7 we make some conclusion remarks.

2 The Dispersion Set

Working with polynomial and rational functions over k we will write $f(x) \perp g(x)$ for $f(x), g(x) \in k[x]$ to indicate that $f(x)$ and $g(x)$ are coprime; if $F(x) \in k(x)$, then $\operatorname{den} F(x)$ is the monic polynomial from $k[x]$ such that $F(x) = \frac{f(x)}{\operatorname{den} F(x)}$ for some $f(x) \in k[x], f(x) \perp \operatorname{den} F(x)$. In this case we write $\operatorname{num} F(x)$ for $f(x)$.

The set of monic irreducible polynomials of $k[x]$ will be denoted by $\mathrm{Irr}(k[x])$. If $p(x) \in \mathrm{Irr}(k[x])$, $f(x) \in k[x]$, then we define the valuation $\mathrm{val}_{p(x)}f(x)$ as the maximal $m \in \mathbb{N}$ such that $p^m(x)|f(x)$ ($\mathrm{val}_{p(x)}0 = \infty$), and $\mathrm{val}_{p(x)}F(x) = \mathrm{val}_{p(x)}(\mathrm{num}\, F(x)) - \mathrm{val}_{p(x)}(\mathrm{den}\, F(x))$ for $F(x) \in k(x)$.

Let $A(x)$ be as in (1), then we define

$$\mathrm{den}\, A(x) = \operatorname*{lcm}_{i=1}^{n} \operatorname*{lcm}_{j=1}^{n} \mathrm{den}(a_{ij}(x)), \;\; \mathrm{den}\, A^{-1}(x) = \operatorname*{lcm}_{i=1}^{n} \operatorname*{lcm}_{j=1}^{n} \mathrm{den}(\tilde{a}_{ij}(x)).$$

If

$$F(x) = (F_1(x), F_2(x), \ldots, F_n(x))^T \in k(x)^n$$

then $\mathrm{den}\, F(x) = \mathrm{lcm}_{i=1}^n \mathrm{den}\, F_i(x)$, and $\mathrm{val}_{p(x)}F(x) = \min_{i=1}^n \mathrm{val}_{p(x)}F_i(x)$. A solution $F(x) = (F_1(x), F_2(x), \ldots, F_n(x))^T \in k(x)^n$ of (1) as well as a solution $F(x) \in k(x)$ of (2), (3) is a *rational* solution. If $\mathrm{den}\, F(x) \neq 1$ then this solution is *non-polynomial*, and *polynomial* otherwise.

If $p(x) \in \mathrm{Irr}(k[x])$, $f(x) \in k[x] \setminus \{0\}$ then we define the finite set

$$\mathcal{N}_{p(x)}(f(x)) = \{m \in \mathbb{Z} \,:\, p(x+m)|f(x)\}. \tag{7}$$

If $\mathcal{N}_{p(x)}(f(x)) = \emptyset$ then define $\max\mathcal{N}_{p(x)}(f(x)) = -\infty$, $\min\mathcal{N}_{p(x)}(f(x)) = +\infty$.

¿From now on we use the notation

$$V(x) = b_n(x-n), \;\; W(x) = b_0(x)$$

for equation (3), and

$$V(x) = u_1(x-1), \;\; W(x) = u_0(x),$$

where $u_1(x) = \mathrm{den}\, A(x)$, $u_0(x) = \mathrm{den}\, A^{-1}(x)$, for system (1).

The first computer algebra algorithm for finding solutions of (3) which belong to $k(x)$ was proposed in [3]. One of the statements proven in [3] (and later in [6] for the case of a system) can be formulated using notation (7) as follows:

Proposition 1. ([3,6]) *Let $p(x)$ divide the denominator of a rational solution of (3) or (1), $p(x) \in \mathrm{Irr}(k[x])$. Then $\max\mathcal{N}_{p(x)}(V(x)) \geq 0$, and $\min\mathcal{N}_{p(x)}(W(x)) \leq 0$.*

For $f(x), g(x) \in k[x] \setminus \{0\}$ we define their *dispersion set*:

$$\mathrm{ds}(f(x), g(x)) = \{h \in \mathbb{N} \,:\, \deg\gcd(f(x), g(x+h)) > 0\} \tag{8}$$

and their *dispersion*:

$$\mathrm{dis}(f(x), g(x)) = \max(\mathrm{ds}(f(x), g(x)) \cup \{-\infty\}). \tag{9}$$

The dispersion is equal to $-\infty$ iff $\deg\gcd(f(x), g(x+h)) = 0$ for all $h \in \mathbb{N}$, and belongs to $\mathbb{N}$ otherwise. The set $\mathrm{ds}(f(x), g(x))$ can be computed as the set of all integer non-negative roots of the polynomial $\mathrm{Res}_x(f(x), g(x+h)) \in k[h]$. This set can be also obtained from the full factorization of $f(x)$ and $g(x)$. Indeed, for given

$w(x), v(x) \in \mathrm{Irr}(k[x])$, $\deg w(x) = \deg v(x) = s$, one can easily recognize whether or not exists $h \in \mathbb{Z}$ such that $w(x+h) = v(x)$: if $w(x) = x^s + w_{s-1}x^{s-1} + \ldots$, $v(x) = x^s + v_{s-1}x^{s-1} + \ldots$, then $w(x+h) = x^s + (w_{m-1} + sh)x^{s-1} + \ldots$ and the only candidate for h is $\frac{v_{s-1} - w_{s-1}}{s}$, if this value belongs to $\mathbb{Z}$ ([20]). The computation is faster if one resorts to the approach from [20] based on the full factorization instead of computing integer roots of a resultant. This is successfully used, e.g., in Maple: LREtools[dispersion].

By Proposition 1, if a non-polynomial rational solution exists then the set $\mathrm{ds}(V(x), W(x))$ is not empty.

3 Algorithms for Constructing Universal Denominators

3.1 The Algorithm $\mathbf{A}_D$ from [5,6]

The algorithm is as follows:

Find $\mathcal{H} = \mathrm{ds}(V(x), W(x))$. If $\mathcal{H} = \emptyset$ then terminate the algorithm with the result $U(x) = 1$ (we suppose below that $\mathcal{H} = \{h_1, h_2, \ldots, h_s\}$ and $h_1 > h_2 > \ldots > h_s$, $s \geq 1$). Set $U(x) = 1$ and successively for $m = 1, 2, \ldots, s$ execute the following group of assignments:

$P(x) = \gcd(V(x), W(x + h_m))$
$V(x) = V(x)/P(x)$
$W(x) = W(x)/P(x - h_m)$
$U(x) = U(x) \prod_{i=0}^{h_m} P(x - i)$.

The final value of $U(x)$ is a universal denominator for equations (2), (3) or, resp., system (1).

We will refer to this algorithm as $\mathbf{A}_D$. This algorithm is exploited in current versions of Maple:

LREtools[ratpolysols], LinearFunctionSystems[UniversalDenominator].

3.2 The Algorithm from [11]

In [11] a more general problem than the search for rational solutions of system (1) was solved. However, the algorithm from [11, Prop. 3] can be used to compute a universal denominator $u(x)$ related to (1). Using our notation (setting in addition $h = \mathrm{dis}(V(x), W(x))$) this algorithm may be represented as follows.

Consider the sequence of polynomials $\{(V_j(x), W_j(x), P_j(x))\}$ defined inductively as:

$$V_0(x) = V(x), \quad W_0(x) = W(x), \quad P_0(x) = \gcd(V(x), W(x+h)),$$

and for $j = 1, 2, \ldots, h$,
$V_j(x) = V_{j-1}(x)/P_{j-1}(x)$,
$W_j(x) = W_{j-1}(x)/P_{j-1}(x - h + j - 1)$,
$P_j(x) = \gcd(V_j(x), W_j(x + h - j))$.

Then

$$u(x) = \prod_{j=0}^{h} \prod_{i=0}^{h-j} P_j(x - i).$$

3.3 The Algorithm $\mathbf{A}_U$ from [14]

An explicit formula for a lower bound of $\mathrm{val}_{p(x)}F(x)$ can be found in [14]: if $F(x)$ is a rational solution of equation (3) or system (1) then

$$\mathrm{val}_{p(x)}F(x) \geq -\min\left\{\sum_{l\in\mathbb{N}} \mathrm{val}_{p(x+l)}V(x),\ \sum_{l\in\mathbb{N}} \mathrm{val}_{p(x-l)}W(x)\right\} \tag{10}$$

for any $p(x) \in \mathrm{Irr}(k[x])$.

This formula was used in [14] as a base for the new algorithm $\mathbf{A}_U$ for computing a universal denominator. This algorithm can be divided into two steps. In the first step, $\mathbf{A}_U$ constructs a finite set M of irreducible polynomials that are candidates for divisors of denominators of rational solutions. At the second step, for each $p(x) \in M$ this algorithm computes the value

$$\gamma_{p(x)} = \min\left\{\sum_{l\in\mathbb{N}} \mathrm{val}_{p(x+l)}V(x),\ \sum_{l\in\mathbb{N}} \mathrm{val}_{p(x-l)}W(x)\right\}. \tag{11}$$

The product $\prod_{p(x)\in M} p^{\gamma_{p(x)}}(x)$ gives a universal denominator related to a given equation or system.

By Proposition 1 we can define

$$M = \left\{p(x) \in \mathrm{Irr}(k[x]) \,:\, \min\mathcal{N}_{p(x)}(W(x)) \leq 0,\ \ \max\mathcal{N}_{p(x)}(V(x)) \geq 0\right\}.$$

For constructing this set the full factorization of polynomials $V(x), W(x)$ has to be found. Then we find the finite set $Q \subset \mathrm{Irr}(k[x])$ such that $q(x) \in Q$ iff

$$\min\mathcal{N}_{q(x)}(W(x)) = 0,\ \ \max\mathcal{N}_{q(x)}(V(x)) \geq 0.$$

Let $Q \neq \emptyset$ and $Q = \{q_1(x), q_2(x), \ldots, q_s(x)\}$, $s \geq 1$. For each $1 \leq i \leq s$ consider

$$M_{q_i(x)} = \{q_i(x), q_i(x+1), \ldots, q_i(x+h_i)\}, \tag{12}$$

where

$$h_i = \max\mathcal{N}_{q_i(x)}(V(x)). \tag{13}$$

We have $M = \bigcup_{i=1}^{s} M_{q_i(x)}$.

3.4 An Improved Version of the Algorithm $\mathbf{A}_U$ (the Algorithm $\mathbf{A}'_U$)

As it is described above the algorithm $\mathbf{A}_U$ contains two steps: the construction of the set M and the computation of $\gamma_{p(x)}$ using (11) for all $p(x) \in M$, which results in the universal denominator. Formula (11) contains the sums by $l \in \mathbb{N}$. In spite of the fact that $\mathbb{N}$ is infinite, the sums have only finite number of summands corresponding to the irreducible factors of $V(x)$ and $W(x)$, which are equal to non-negative and non-positive shifts of $p(x)$, respectfully (the corresponding valuations are equal to the exponents of such factors in the factorization of $V(x)$

and $W(x)$). No special way for time saving computing of the exponents $\gamma_{p(x)}$ was described in [14]. We propose below a possible way of this kind.

It is clear that when we compute (11) for $p(x) = q_i(x+j) \in M_{q_i(x)}$ (where $M_{q_i(x)}$ is as in (12)), the corresponding $\gamma_{q_i(x+j)}$ might be equal for many successive j. Indeed if we have computed $\gamma_{q_i(x)}$, and after that we compute $\gamma_{q_i(x+j)}$ for j from 1 to h_i, then the value can be changed only for those j for which there is an irreducible factor of $V(x)$ and/or $W(x)$ equal to $q_i(x+j)$ (such *critical* points can be computed in advance while constructing the set M). The consideration is a basis for the improved version of the algorithm $\mathbf{A}_U$; the new algorithm is presented below in details.

The first step is adjusted to compute the following:

- $\{q_i(x)\}_{i=1}^s$ and $\{h_i\}_{i=1}^s$ which correspond (12) and (13).
- The sets
$$C^{i,W} = \left\{c \in \mathbb{Z} : \mathrm{val}_{q_i(x+c)}(W(x)) > 0\right\},$$
$$C^{i,V} = \left\{c \in \mathbb{Z} : \mathrm{val}_{q_i(x+c-1)}(V(x)) > 0\right\}$$
of the critical points.
- $D^{i,W} = \left\{D_c^{i,W}\right\}$ and $D^{i,V} = \left\{D_c^{i,V}\right\}$ which are the sets of the valuations corresponding to the critical points: $D_c^{i,W} = \mathrm{val}_{q_i(x+c)}(W(x))$ for each $c \in C^{i,W}$ and $D_c^{i,V} = \mathrm{val}_{q_i(x+c-1)}(V(x))$ for each $c \in C^{i,V}$.

Note that all the data are computed simultaneously using the factorizations of $W(x)$ and $V(x)$.

The second step is performed as a loop by i from 1 to s. For each $q(x) = q_i(x)$ and $h = h_i$ execute the following:

- Construct the joint and sorted set of critical points:
$\left\{c_j : c_j \geq 0, c_j \leq h, c_j \in C^{i,W} \bigcup C^{i,V}\right\}_{j=1}^{n_i}$, with $c_1 < c_2 < \ldots < c_{n_i}$.
- Compute the intervals $\{l_0, \ldots, l_1 - 1\}$, $\{l_1, \ldots, l_2 - 1\}$, $\ldots$ $\{l_{k-1}, \ldots, h\}$ of the same exponents γ_1, γ_2, …, γ_k and the exponents themselves:
We initialize the computation with $k=0$, $\gamma_0 = -1$ and $\gamma_w = \sum_{0>c\in C^{i,W}} D_c^{i,W}$, $\gamma_v = \sum_{0\geq c\in C^{i,V}} D_c^{i,V}$. Then for the critical points $c = c_1, c_2, \ldots c_{n_i}$ we compute the change of the values by $\gamma_w = \gamma_w + D_c^{i,W}$ (if $c \in C^{i,W}$) and/or $\gamma_v = \gamma_v - D_c^{i,V}$ (if $c \in C^{i,V}$), which gives a new $\gamma_{k+1} = \min(\gamma_v, \gamma_w)$. If $\gamma_{k+1} \neq \gamma_k$ then a new interval with the new exponent γ_{k+1} is started from $l_k = c$ (after that k is correspondingly increased by 1).
- Having added $l_k = h + 1$, compute the factor of the universal denominator that corresponds $q(x)$:
$U_i = \prod_{m=1}^{k} \prod_{j=l_{m-1}}^{l_m - 1} q(x+j)^{\gamma_m}$

The final universal denominator is the product of all U_i for $i = 1, 2, \ldots s$.

The algorithm is justified by considering the changes in (11) for computing $\gamma_{q(x+j)}$ with successive j. Note that γ_v (i.e. $\sum_{l\in \mathbb{N}} \mathrm{val}_{q(x+j+l)} V(x)$) and γ_w (i.e. $\sum_{l\in \mathbb{N}} \mathrm{val}_{q(x+j-l)} W(x)$) change a bit differently with the increase of j: the first one may only decrease and the second one may only increase. It leads to the corresponding differences in the algorithm in the definitions of $C^{i,W}$, $C^{i,V}$ and the formulas for the initial values of γ_w, γ_v and their changes.

We will refer to this detailed (improved) version of $\mathbf{A}_U$ as $\mathbf{A}'_U$.

4 Analysis of the Algorithms

4.1 Equivalence of Results

Proposition 2. *The universal denominators computed by the algorithms described in Section 3.2 coincide for any given $V(x), W(x)$. Intermediate polynomials computed by $\mathbf{A}_D$ are also computed as intermediate polynomials by the algorithm from [11].*

Proof. First show that the algorithm from [11] gives the same result and computes all the intermediate polynomials that $\mathbf{A}_D$ computes. Indeed, replace $\mathcal{H}$ by

$$\bar{\mathcal{H}} = \{h, h-1, \ldots, 0\}$$

$h = h_1 = \operatorname{dis}(V(x), W(x))$. This extension of $\mathcal{H}$ does not change the result (the additionally computed gcd's will be equal to 1). We also enumerate the values $V(x), W(x), P(x), U(x)$ in $\mathbf{A}_D$:

Set $U_0(x) = 1, V_0(x) = V(x), W_0(x) = W(x)$ and successively for $j = 0, 1, \ldots,$ $h-1$ execute the following group of assignments:

$P_{j+1}(x) = \gcd(V_j(x), W(x+h-j))$
$V_{j+1}(x) = V_j(x)/P_{j+1}(x)$
$W_{j+1}(x) = W_j(x)/P_{j+1}(x-h+j)$
$U_{j+1}(x) = U_j(x)\prod_{i=0}^{h-j} P_{j+1}(x-i)$.

Evidently triples $(V_t(x), W_t(x), P_t(x))$ coincide for $t = 0, 1, \ldots h$ in both algorithms, and $u(x) = U_h(x)$.

It was proven in [11] that if $h = \operatorname{dis}(V(x), W(x))$ then

$$u(x) = \gcd\left(\prod_{i=0}^{h} V(x-i),\ \prod_{i=0}^{h} W(x+i)\right).$$

Therefore, the value $\operatorname{val}_{p(x)}u(x)$ is equal to the right-hand side of (11) for any $p(x) \in \operatorname{Irr}(k[x])$. This implies that the outputs of the algorithm from 3.2 and $\mathbf{A}_U$ coincide. Thus, the outputs of $\mathbf{A}_D$ and, resp. $\mathbf{A}_U$ coincide as well. The coincidence of the outputs of $\mathbf{A}_U$ and $\mathbf{A}'_U$ is evident. □

4.2 Complexity Comparison

We now give a complexity analysis of $\mathbf{A}_D$ and $\mathbf{A}'_U$. Let $n = \max\{\deg V(x), \deg W(x)\}$ and $h = \operatorname{dis}(V(x), W(x))$. We compare the complexities $T_D(n,h)$ and $T_U(n,h)$ of $\mathbf{A}_D$ and $\mathbf{A}'_U$. In this context, the complexity is the number of the field operations in k in the worst case.

Both algorithms perform polynomial multiplications for getting $U(x)$. We do not specify the used polynomial multiplication algorithm, but suppose that the worst case is when it is necessary to multiply a big number (which is equal to $\deg U(x)$) of first degree polynomials.

Both algorithms spend the same time to find the full factorization of $V(x)$ and $W(x)$ and to compute their dispersion set. In addition, $\mathbf{A}'_U$ constructs the set Q as well as the set of corresponding h_i, the set of critical points, and the set of corresponding valuations. The cost of this computation in the worst case is $O(n)$ plus the cost of sorting critical points. This gives totally $O(n \log n)$.

On the other hand, $\mathbf{A}_D$ computes gcd's; if $h \geq n$ then in the worst case, the cost of this computation is $\sum_{i=0}^{n} T_{\mathrm{gcd}}(n-i)$, where $T_{\mathrm{gcd}}(n)$ is the complexity of the gcd computation for two polynomials whose maximal degree is n. If $0 < h < n$ then the cost in the worst case is $\sum_{i=0}^{h} T_{\mathrm{gcd}}(n-i)$. Obviously $\sum_{i=0}^{n} T_{\mathrm{gcd}}(n-i) = \sum_{i=0}^{n} T_{\mathrm{gcd}}(i)$, $\sum_{i=0}^{h} T_{\mathrm{gcd}}(n-i) = \sum_{i=n-h}^{n} T_{\mathrm{gcd}}(i)$, and we have the following proposition.

Proposition 3. *If $T_{\mathrm{gcd}}(n)/(n \log n) \to \infty$ then the difference $T_D(n,h)$–$T_U(n,h)$ is positive for almost all $n, h \in \mathbb{N}^+$ and*

$$T_D(n,h) - T_U(n,h) = \begin{cases} \sum_{i=0}^{n} T_{\mathrm{gcd}}(i) + O(n \log n), & \text{if } h \geq n, \\ \sum_{i=n-h}^{n} T_{\mathrm{gcd}}(i) + O(n \log n), & \text{if } h < n. \end{cases} \tag{14}$$

In the next proposition we use the Ω-notation which is very common in complexity theory ([19]). Unlike O-notation which is used for describing upper asymptotical bounds, the Ω-notation is used for describing lower asymptotical bounds.

Proposition 4. *Let $T_{\mathrm{gcd}}(n) = \Omega(n^d)$, $d > 1$. Then the difference $T_D(n,h) - T_U(n,h)$ is positive almost all $n, h \in \mathbb{N}^+$ and is $\Omega(R(n,h))$, where*

$$R(n,h) = \begin{cases} n^{d+1}, & \text{if } h \geq n, \\ hn^d, & \text{if } h < n. \end{cases}$$

Proof. The case $h \geq n$ follows from (14) and $d > 1$. In the case $h < n$ we can use the inequality

$$\sum_{i=m}^{n} i^d > \frac{n^d (n-m)}{d+1} \tag{15}$$

which is valid for any integer $0 < m \leq n$ and real $d \geq 1$. Taking $m = n-h$ we get the claimed. To prove (15) note that the function x^d is monotonically increasing when $x \geq 0$ and $d \geq 1$. This gives for $m < n$ (the case $m = n$ is trivial):

$$\sum_{i=m}^{n} i^d > \sum_{i=m+1}^{n} i^d > \sum_{i=m+1}^{n} \int_{i-1}^{i} x^d \,\mathrm{d}x = \int_{m}^{n} x^d \,\mathrm{d}x = \frac{n^{d+1}}{d+1} \left(1 - \left(\frac{m}{n}\right)^{d+1}\right).$$

Since in our case $1 - \left(\frac{m}{n}\right)^{d+1} \geq 1 - \frac{m}{n}$, we get (15). $\square$

To the authors' knowledge $T_{\mathrm{gcd}}(n) = \Omega(n^d)$, $d > 1$, for the algorithms now in use in actual practice for gcd computations.

The fast Euclidean algorithm [12, Ch. 11] has complexity $O(n \log^2 n \log\log n)$ if Fast Fourier Transform is used to multiply polynomials. But this version of the fast Euclidean algorithm is not practical due to a big constant hidden in O. Nevertheless, if we suppose that the fast Euclidean algorithm is used and the estimate $\Omega(n\ \log^2 n\ \log\log n)$ (or, even $\Omega(n\ \log^2 n)$) is valid for the complexity of this algorithm then by Proposition 3 the difference $T_D(n,h) - T_U(n,h)$ is positive (i.e., $T_U(n,h) < T_D(n,h)$) for almost all $n, h \in \mathbb{N}^+$.

5 Implementation

Below we consider an implementation in Maple of $\mathbf{A}'_U$ (Section 5) and demonstrate the corresponding time comparison with $\mathbf{A}_D$ (Section 6). As it was shown in Proposition 2 both algorithms give the same result, and the comparison is correct. The algorithm from [11] is similar to $\mathbf{A}_D$ by Proposition 2, and we do not involve this algorithm into the comparison.

As we mentioned in Section 3.1 the algorithm $\mathbf{A}_D$ implementation is available in Maple as an internal procedure of the package LREtools. We implemented our new algorithm $\mathbf{A}'_U$ and performed experimental comparison of the two algorithms.

The implementation has several peculiarities which are discussed below.

5.1 Full Factorization

The algorithm $\mathbf{A}'_U$ (and $\mathbf{A}_U$ as well) is based on the full factorization of the given polynomials $V(x)$ and $W(x)$. Our implementation uses the result of the factorization not only to construct the set M of irreducible polynomials, but also computes (11) using it. Note that it is not the case for the implementation of the algorithm $\mathbf{A}_D$ in Maple. It uses the procedure LREtools[dispersion] to compute the dispersion of polynomials which implements the algorithm [20], i.e., uses the full factorization. But the next steps of the algorithm $\mathbf{A}_D$ are implemented as presented not exploiting the result of the factorization of the previous step.

5.2 Shift Computation

Our implementation uses vastly the auxiliary procedure, which given $p(x), r(x) \in \mathrm{Irr}(k[x])$ computes the shift $s \in \mathbb{Z}$ such that $p(x) = r(x+s)$ or defines that no such s exists (actually it is a particular case of computing $\mathcal{N}_{p(x)}(r(x))$ when $r(x) \in \mathrm{Irr}(k[x])$). The procedure is used both to compute the set M and to compute $C^{i,W}$, $C^{i,V}$, $D^{i,W}$, $D^{i,V}$ for further computations of the exponents γ. The shift computation is implemented efficiently using the main idea of the algorithm [20] presented in the end of Section 2.

5.3 Computing Universal Denominator

Though we compute the values γ successively, it is better to compute the universal denominator at once for all $q_i(x+j)$, rather than compute the universal denominator also successively. In the latter case, the intermediate computations of the preliminary results might be costly at least in Maple.

6 Three Experiments

Using our implementation of the algorithm $\mathbf{A}'_U$ and the implementation of the algorithms $\mathbf{A}_D$ that is embedded in Maple, we have performed three experiments to compare the algorithms.

6.1 Experiment 1

We have applied both algorithms to the following similar inputs:

(a) $V(x) = W(x) = \prod_{i=1}^{l}(x + m + 1/i)(x - m + 1/i)$ for $m = 20, 100, 500, 2500$, $l = 1, 15, 30, 45, 60$;
(b) $V(x)=W(x)=\prod_{i=1}^{l}(x+m+i+1/i)(x-m-i+1/i)$ for $m=20, 100, 500, 2500$, $l = 1, 15, 30, 45, 60$.

The corresponding universal denominators found by both algorithm for the inputs are, respectfully, the following:

(a) $\prod_{i=1}^{l}\prod_{j=-m}^{m}(x - j - 1 + 1/i)$;
(b) $\prod_{i=1}^{l}\prod_{j=-m-i}^{m+i}(x - j + 1/i)$.

The experiment is based on the example from [15], which is transformed to be more complicated by using l similar pair factors instead of the only one pair. Tables 1 and 2 show the CPU time[1] needed to compute the corresponding universal denominators by three implementations for each of the pair $V(x)$ and $W(x)$. The input polynomials are expanded before calling the implementations. The expansion is needed to create equal conditions for both algorithms (otherwise the factored input definitely simplifies the work for $\mathbf{A}'_U$). In addition, it has been found that the implementation of the algorithm [20] in Maple for computing the dispersion of two polynomials uses some additional preprocessing which leads to inefficiency for the inputs in the factored form at least in our experiments, so the expanded input allows eliminating this question in our comparison.

Table 1. Results of the experiment 1(a), in seconds

	m=20		m=100		m=500		m=2500	
	$\mathbf{A}'_U$	$\mathbf{A}_D$	$\mathbf{A}'_U$	$\mathbf{A}_D$	$\mathbf{A}'_U$	$\mathbf{A}_D$	$\mathbf{A}'_U$	$\mathbf{A}_D$
l=1	0.000	0.016	0.000	0.015	0.015	0.015	0.016	0.031
l=15	0.079	0.141	0.094	0.141	0.172	0.203	0.546	0.438
l=30	0.375	0.547	0.407	0.562	0.547	0.656	1.266	1.109
l=45	0.719	1.140	0.828	1.235	1.172	1.531	3.015	2.531
l=60	2.032	2.875	2.390	3.344	3.000	4.516	5.063	5.704

The results show that the algorithms behave differently with the growth of m and l. The results of $\mathbf{A}_D$ are getting relatively worse with the growth of l if we fix any m, and the results of $\mathbf{A}'_U$ are getting relatively worse with the growth of m if

[1] For all the experiments: Maple 13, Windows XP, Pentium 4 1.7 GHz, 512 MB RAM.

Table 2. Results of the experiment 1(b), in seconds

	m=20		m=100		m=500		m=2500	
	$\mathbf{A}'_U$	$\mathbf{A}_D$	$\mathbf{A}'_U$	$\mathbf{A}_D$	$\mathbf{A}'_U$	$\mathbf{A}_D$	$\mathbf{A}'_U$	$\mathbf{A}_D$
l=1	0.016	0.015	0.000	0.000	0.000	0.016	0.031	0.031
l=15	0.078	0.375	0.109	0.422	0.172	0.531	0.578	1.032
l=30	0.359	2.890	0.407	3.063	0.531	3.484	1.266	5.344
l=45	0.860	10.641	0.796	11.547	1.516	13.234	3.078	17.656
l=60	2.406	31.187	2.719	33.484	2.657	37.125	4.766	44.797

we fix any l. The latter observation may be explained if we analyze the structure of the algorithms in respect to the particular problem in hand: actually, for the fixed m and given l $\mathbf{A}'_U$ performs similar set of operations l times, but $\mathbf{A}_D$ needs to perform gcd computations with the polynomials of l times higher degrees.

It is easy to see that the inputs (a) are more convenient for the algorithm $\mathbf{A}_D$: the gcd is computed only once for each input, the number of multiplied polynomials is $2m + 1$, while for $\mathbf{A}'_U$ this number is $(2m + 1)l$. In spite of this handicap the timing of $\mathbf{A}'_U$ looks better for the whole experiment (Table 1).

The input (b) corresponds near to the worst case for both algorithms $\mathbf{A}'_U$ and $\mathbf{A}_D$ (the input size is a pair of numbers as in Section 4.2), and an advantage of $\mathbf{A}'_U$ is evident (Table 2).

We have noted in Section 3.4 that no special way for time saving computing of the exponents $\gamma_{p(x)}$ was proposed in the description of $\mathbf{A}_U$ given in [14]. If one uses formula (11) for each $p(x) \in M$ then the total computation time increases dramatically. We have implemented a straightforward version of $\mathbf{A}_U$ as well for the preliminary experiments and, for example, the result of $\mathbf{A}_U$ for the input of type (a) with $m = 2500$, $l = 60$ is 350 seconds.

6.2 Experiment 2

We have also applied the algorithms to several sets of randomly generated pairs of polynomials $V(x)$ and $W(x)$. Each set contains 500 pairs, and each polynomial is generated using Maple command randpoly(x,degree=d,terms=l), i.e., it is a polynomial of degree up to d and it contains up to l terms. Note that given such generated polynomials, the universal denominator found by the considered algorithms is most probably x^n for some $n \in \mathbb{N}$, and moreover it is just 1 for most of the cases. Still the experiment is meaningful, since if we try to search for the rational solution of absolutely arbitrary equations, it would be exactly like this. 9 sets are generated for $d = 10, 20, 30$ and $l = 2, d/2, d$. Table 3 shows the CPU time needed to compute the corresponding universal denominators by the implementations for each of the set. We do not need to expand the input polynomials before calling the implementations in the experiment since the polynomials are expanded by construction.

The results show that $\mathbf{A}'_U$ is better than $\mathbf{A}_D$ in this experiment for all the sets.

Table 3. Results of the experiment 2, in seconds

	l=2		l=d/2		l=d	
	$\mathbf{A}'_U$	$\mathbf{A}_D$	$\mathbf{A}'_U$	$\mathbf{A}_D$	$\mathbf{A}'_U$	$\mathbf{A}_D$
d=10	1.578	6.016	5.953	9.578	6.734	10.157
d=20	1.750	8.094	8.594	12.938	9.828	13.969
d=30	1.922	10.422	12.235	17.234	13.985	19.375

6.3 Experiment 3

We have also applied the algorithms to several sets of other randomly generated pairs of polynomials $V(x)$ and $W(x)$. Each set contains again 500 pairs, but the polynomials are generated differently. Each polynomial is generated as a product of at most l factors of the form $(x - r_i)^{d_i}$, where r_i is a random integer between -10 and 10, d_i is a random integer between 0 and d. Such method of generation ensures that the found universal denominators will be non trivial. 9 sets are generated for $d = 2, 4, 6$ and $l = 1, 5, 10$. Table 4 shows the CPU time needed to compute the corresponding universal denominators by the implementations for each of the set. The input polynomials are expanded before calling the implementations.

Table 4. Results of the experiment 3, in seconds

	l=1		l=5		l=10	
	$\mathbf{A}'_U$	$\mathbf{A}_D$	$\mathbf{A}'_U$	$\mathbf{A}_D$	$\mathbf{A}'_U$	$\mathbf{A}_D$
d=1	0.219	0.890	1.094	2.453	2.672	6.265
d=3	0.390	1.390	2.953	6.500	5.844	14.937
d=5	0.437	1.609	4.328	9.750	8.313	23.250

The results show that $\mathbf{A}'_U$ is better than $\mathbf{A}_D$ in this experiment for all the sets.

7 Conclusion

Our investigation presented in the paper has confirmed that it might be useful to revisit the problems which were solved earlier by the algorithms which use incomplete factorization based on computation of the greatest common divisors as a result of the desire to avoid the use of the full factorization. The full factorization based algorithm $\mathbf{A}_U$ (and its new improved version $\mathbf{A}'_U$) for the universal denominator construction is proved to deliver the same results as the old gcd computation based algorithm $\mathbf{A}_D$, but the implementation of $\mathbf{A}'_U$ is shown to be more efficient. It is especially logical to switch to the new approach, in particular, in Maple, since the existing Maple implementation of the algorithm $\mathbf{A}_D$ uses already the factorization based auxiliary algorithm for computing the dispersion, i.e., the required factorizations are computed already.

Note that new algorithms should not be necessary obtained out of the old ones just by substituting gcd computations with the corresponding computations using the results of factorizations. It might be more useful to re-think the whole algorithm over again based on the new approach. In this way, $\mathbf{A}'_U$ utilizes the new computations based on the new formula (11), and they are implemented efficiently, e.g., taking into account the fact that when we compute (11) for $p(x) = q_i(x + j) \in M_{q_i(x)}$ from (12), the corresponding $\gamma_{q_i(x+j)}$ might be equal for many successive j.

Logically if the basic operations are significantly changed then concepts for algorithms designing have to be updated.

Acknowledgments. The authors are grateful to M. Barkatou and M. Petkovšek for interesting discussions, and to anonymous referees for their helpful comments.

References

1. Abramov, S.: On the summation of rational functions. USSR Comput. Math. Phys. 11, 324–330 (1971); Transl. from Zh. vychisl. mat. mat. fyz. 11, 1071–1075 (1971)
2. Abramov, S.: Problems of computer algebra involved in the search for polynomial solutions of linear differential and difference equations. Moscow Univ. Comput. Math. Cybernet. 3, 63–68 (1989); Transl. from Vestn. MGU. Ser. 15. Vychisl. mat. i kibernet. 3, 53–60 (1989)
3. Abramov, S.: Rational solutions of linear difference and differential equations with polynomial coefficients. USSR Comput. Math. Phys. 29, 7–12 (1989); Transl. from Zh. vychisl. mat. mat. fyz. 29, 1611–1620 (1989)
4. Abramov, S.: Rational solutions of linear difference and q-difference equations with polynomial coefficients. In: ISSAC 1998 Proceedings, pp. 303–308 (1995)
5. Abramov, S.: Rational solutions of linear difference and q-difference equations with polynomial coefficients. Programming and Comput. Software 21, 273–278 (1995); Transl. from Programmirovanie 6, 3–11 (1995)
6. Abramov, S., Barkatou, M.: Rational solutions of first order linear difference systems. In: ISSAC 1998 Proceedings, pp. 124–131 (1998)
7. Abramov, S., Bronstein, M., Petkovšek, M.: On polynomial solutions of linear operator equations. In: ISSAC 1995 Proceedings, pp. 290–295 (1995)
8. Abramov, S., van Hoeij, M.: A method for the integration of solutions of Ore equations. In: ISSAC 1997 Proceedings, pp. 172–175 (1997)
9. Abramov, S., van Hoeij, M.: Integration of solutions of linear functional equations. Integral Transforms and Special Functions 8, 3–12 (1999)
10. Abramov, S., Ryabenko, A.: Indicial rational functions of linear ordinary differential equations with polynomial coefficients. Fundamental and Applied Mathematics 14(4), 15–34 (2008); Transl. from Fundamentalnaya i Prikladnaya Matematika 14(4), 15–34 (2008)
11. Barkatou, M.: Rational solutions of matrix difference equations: problem of equivalence and factorization. In: ISSAC 1999 Proceedings, pp. 277–282 (1999)
12. Von zur Gathen, J., Gerhard, J.: Modern Computer Algebra, 2nd edn. Cambridge University Press, Cambridge (2003)
13. Gerhard, J.: Modular Algorithms in Symbolic Summation and Symbolic Integration. LNCS, vol. 3218. Springer, Heidelberg (2004)

14. Gheffar, A., Abramov, S.: Valuations of rational solutions of linear difference equations at irreducible polynomials. Adv. in Appl. Maths (submitted 2010)
15. van Hoeij, M.: Rational solutions of linear difference equations. In: ISSAC 1998 Proceedings, pp. 120–123 (1998)
16. van Hoeij, M.: Factoring polynomials and the knapsack problem. J. Number Theory 95, 167–189 (2002)
17. van Hoeij, M., Levy, G.: Liouvillian solutions of irreducible second order linear difference equations. In: ISSAC 2010 Proc. (2010)
18. Khmelnov, D.E.: Search for polynomial solutions of linear functional systems by means of induced recurrences. Programming and Comput. Software 30, 61–67 (2004); Transl. from Programmirovanie 2, 8–16 (2004)
19. Knuth, D.E.: Big omicron and big omega and big theta. ACM SIGACT News 8(2), 18–23 (1976)
20. Man, Y.K., Wright, F.J.: Fast polynomial dispersion computation and its application to indefinite summation. In: ISSAC 1994 Proceedings, pp. 175–180 (1994)
21. Petkovšek, M.: Hypergeometric solutions of linear recurrences with polynomial coefficients. J. Symbolic Computation 14, 243–264 (1992)
22. Maple online help, `http://www.maplesoft.com/support/help/`