Irreducibility Testing of Lacunary 0,1-Polynomials

Michael Filaseta[1]
Mathematics Department
University of South Carolina
Columbia, SC 29208

Douglas B. Meade
Mathematics Department
University of South Carolina
Columbia, SC 29208

February 6, 2001

[1]The first author was supported by grants from the National Security Agency.

1 Introduction

For $w(x) \in \mathbb{C}[x]$ with $w(x) \not\equiv 0$, define the reciprocal of $w(x)$ to be the polynomial

$$\widetilde{w}(x) = x^{\deg w} w(1/x).$$

We refer to $w(x)$ as being reciprocal if $w(x) = \pm\widetilde{w}(x)$ and as being non-reciprocal otherwise. Irreducibility will refer to irreducibility over the integers (so that, in particular, ± 1 are neither reducible nor irreducible). We define the non-reciprocal part of a monic polynomial $w(x)$ in $\mathbb{Z}[x]$ to be $w(x)$ with its monic reciprocal irreducible factors removed. The purpose of this paper is to establish the following result:

Theorem 1. *There is an algorithm for determining whether the non-reciprocal part of* $f(x) = \sum_{j=0}^{r} x^{d_j}$, *with* $0 = d_0 < d_1 < d_2 < \cdots < d_r = n$, $r \geq 2$, *and* $n \geq 2$, *is irreducible that runs in time* $\ll 2^r r \log r \log n$. *Furthermore, if the non-reciprocal part of* $f(x)$ *is reducible, then the algorithm determines a non-trivial factor of* $f(x)$ *expressed in the form* $\gcd(f(x), w(x))$ *where* $w(x) = \sum_{j=0}^{r} x^{k_j} \in \mathbb{Z}[x]$ *for some integers* k_j *satisfying* $0 = k_0 < k_1 < k_2 < \cdots < k_r = n$.

If r is small compared to n, we say that $f(x)$ is lacunary (a precise definition is unnecessary here). The significance of the result above is that typically such an algorithm can be used to determine if a polynomial $f(x)$ is irreducible by checking whether the following conditions hold:

(i) The polynomial $f(x)$ is itself non-reciprocal.

(ii) The non-reciprocal part of $f(x)$ is irreducible.

(iii) The greatest common divisor of $f(x)$ and $\tilde{f}(x)$ is 1.

For "most" polynomials $f(x)$, condition (i) will hold. If (i) holds and either (ii) or (iii) does not hold, then $f(x)$ is reducible. If, however, conditions (i), (ii), and (iii) all hold, then $f(x)$ is irreducible. The condition (i) is quickly checked. Theorem 1 implies that for a fixed number of non-zero terms of an $f(x)$, with coefficients all 0 and 1, one can check whether (ii) holds in time $\mathcal{O}(\log \deg f)$. To determine the irreducibility of a non-reciprocal $f(x)$ with coefficients all 0 and 1, it remains to determine whether condition (iii) holds. To our knowledge, determining if $\gcd(f(x), \tilde{f}(x)) = 1$ requires considerably more time than $\mathcal{O}(\log \deg f)$. Nevertheless, the above does give a convenient method for determining whether a lacunary polynomial $f(x)$ with coefficients all 0 and 1 is irreducible. A web-based implementation of this irreducibility test, making use of the algorithm in Theorem 1 for (ii), currently can be found at:

http://www.math.sc.edu/~filaseta/irreduc.html

2 Preliminary Remarks Concerning 0,1-Polynomials

We will make use of two preliminary results concerning $0, 1$-polynomials. Define

$$S = \left\{ \sum_{j=0}^{t} \epsilon_j x^j : t \in \mathbb{Z}^+, \epsilon_j \in \{0, 1\} \text{ for each } j, \epsilon_0 = 1 \right\}$$

and

$$S_r = \left\{ \sum_{j=0}^{r} x^{a_j} : a_j \in \mathbb{Z}, 0 = a_0 < a_1 < a_2 < \cdots < a_r \right\}.$$

Thus, S is the set of all polynomials $f(x)$ having each coefficient either 0 or 1 and satisfying $f(0) \neq 0$ and S_r is the subset of such polynomials having exactly $r+1$ terms.

Lemma 1. *Let r be a positive integer, and let $f(x) \in S_r$. Then the non-reciprocal part of $f(x)$ is reducible if and only if there is a $w(x) \in S_r$ satisfying $w(x) \neq f(x)$, $w(x) \neq \tilde{f}(x)$, and $w(x)\widetilde{w}(x) = f(x)\tilde{f}(x)$.*

The proof of the above lemma can be found in [1]. To explain our next lemma, we note that it is possible for a reciprocal polynomial to have non-reciprocal factors or to consist entirely of non-reciprocal irreducible factors. For example,

$$f(x) = (x^3 + x + 1)(x^3 + x^2 + 1) = x^6 + x^5 + x^4 + 3x^3 + x^2 + x + 1$$

has this property. Note that the non-reciprocal part of $f(x)$ is reciprocal (and in fact $f(x)$) for this example. On the other hand, $f(x) \notin S$. For polynomials in S, the situation is different.

Lemma 2. *Let $f(x)$ be a reciprocal polynomial in S. Then $f(x)$ is not divisible by a non-reciprocal polynomial in $\mathbb{Z}[x]$.*

We prove Lemma 2 by contradiction. Let $f(x)$ be a reciprocal polynomial in S, and assume $f(x)$ is divisible by a non-reciprocal polynomial in $\mathbb{Z}[x]$. Then $f(x)$ is divisible by a non-reciprocal irreducible $g(x) \in \mathbb{Z}[x]$. Suppose that $g(x)$ is monic (otherwise, consider $-g(x)$). Observe that $g(0) > 0$ since otherwise $g(x)$ being monic would imply $g(x)$ has a nonnegative real root contradicting that $g(x)$ is a factor of $f(x)$ which has only nonnegative coefficients (and $f(0) \neq 0$). Now, $g(0)|f(0)$ and $f(0) = 1$ implies that $g(0) = 1$. Thus, $g(x)$ is monic and has constant term 1. Let α be a root of $g(x)$ and, hence, $f(x)$. Since $f(x)$ is reciprocal, it follows that $1/\alpha$ is a root of $f(x)$. On the other hand, $\tilde{g}(x)$ is a monic irreducible polynomial that has $1/\alpha$ as a root. It follows that $\tilde{g}(x)$ (the minimal polynomial for $1/\alpha$) divides $f(x)$. Furthermore, $\tilde{g}(x)$ is non-reciprocal because $g(x)$ is. Clearly, $\tilde{g}(x) \neq g(x)$. Therefore, we deduce that the non-reciprocal part of $f(x)$ is reducible.

We apply Lemma 1, and write

$$f(x) = \sum_{j=0}^{r} x^{d_j} \quad \text{and} \quad w(x) = \sum_{j=0}^{r} x^{k_j} \tag{1}$$

where $0 = d_0 < d_1 < \cdots < d_r$ and $0 = k_0 < k_1 < \cdots < k_r$. Thus, $w(x)$ is different from $f(x) = \tilde{f}(x)$ and such that

$$w(x)\widetilde{w}(x) = f(x)\tilde{f}(x). \tag{2}$$

Let $n = \deg f$, and it follows from (2) that $k_r = d_r = n$. Expand the products on each side of (2). The exponents on the left will be of the form $n - k_j + k_i$ where i and j are in $\{0, 1, \ldots, r\}$, and the exponents on the right will be of the form $n - d_j + d_i$ where i and j are in $\{0, 1, \ldots, r\}$.

We continue by induction. Observe that $k_0 = d_0$ and $k_r = d_r$. Suppose that $k_j = d_j$ and $k_{r-j} = d_{r-j}$ for all nonnegative integers j less than some positive integer $t \leq r/2$. We prove $k_t = d_t$ and $k_{r-t} = d_{r-t}$. Consider (2) again with both sides expanded. Compare the exponents $n - k_j + k_i$ on the left with the exponents $n - d_j + d_i$ on the right. The induction hypothesis implies that $n - k_j + k_i = n - d_j + d_i$ if each of i and j is in

$$T = \{0, 1, \ldots, t-1\} \cup \{r, r-1, \ldots, r-t+1\}.$$

Eliminate the terms $x^{n-k_j+k_i}$ on the left and the terms $x^{n-d_j+d_i}$ on the right for such i and j. Consider the uncanceled terms in (2). Denote the remaining set of exponents $n - k_j + k_i$ on the left by L and the remaining set of exponents $n - d_j + d_i$ on the right by R. In other words,

$$L = \{n - k_j + k_i : i \notin T \text{ or } j \notin T\} \quad \text{and} \quad R = \{n - d_j + d_i : i \notin T \text{ or } j \notin T\}.$$

By (2), $L = R$. Consider the least element from each set. For R, we want $n - d_j + d_i$ as small as possible with at least one of i or j not in T. Given the ordering of the d_j's, this minimum is achieved by $n - d_r + d_t = d_t$ or $n - d_{r-t} + d_0 = n - d_{r-t}$. Since $f(x)$ is reciprocal, we deduce that $d_t = n - d_{r-t}$. Thus, d_t is the minimal element in R and it occurs as an exponent at least twice on the right-hand side of (2). In fact, since $n - d_r + d_t$ increases if the subscript t is replaced with a larger subscript and since $n - d_{r-t} + d_0$ increases if the subscript $r - t$ is replaced with a smaller subscript, we deduce that among the uncanceled terms in (2) on the right, the exponent d_t occurs exactly twice. Similarly, one of $n - k_r + k_t = k_t$ and $n - k_{r-t} + k_0 = n - k_{r-t}$ is the minimal element of L. We do not know that $w(x)$ is reciprocal, so we cannot immediately deduce $k_t = n - k_{r-t}$. However, the least exponent among the uncanceled terms in (2) on the left must equal the least exponent among the uncanceled terms in (2) on the right and both must occur as an exponent an equal number of times on their respective sides. This can only occur if $k_t = n - k_{r-t} = d_t = n - d_{r-t}$. We deduce that $k_t = d_t$ and $k_{r-t} = d_{r-t}$. This completes the induction.

It easily follows now that $w(x) = f(x)$. This is a contradiction. Hence, the lemma follows.

Lemma 2 implies that if our input $f(x) \in S$ is reciprocal, then there is no need to do further work to determine the irreducibility of the non-reciprocal part of $f(x)$. In this case, the non-reciprocal part of $f(x)$ is 1. As noted earlier, 1 is neither reducible nor irreducible and so the non-reciprocal part of $f(x)$ is not irreducible. Also, note that the converse of Lemma 2 holds as well. In other words, if $f(x)$ is not divisible by a non-reciprocal polynomial (that is if the non-reciprocal part of $f(x)$ is 1), then $f(x)$ is reciprocal. This follows since a product of reciprocal polynomials is reciprocal.

3 The Factoring Tree

In the next section, we describe an algorithm for determining whether the non-reciprocal part of a given polynomial $f(x) \in S$ is irreducible. The idea behind the algorithm is to determine, from a given $f(x) \in S_r$, all possible $w(x) \in S_r$ for which (2) holds. We do this by constructing a tree whose nodes (or vertices) are pairs (A, B) with A being a list of exponents

k_j appearing in such a $w(x)$ and B being a list of exponents $n-k_j$ appearing in $\widetilde{w}(x)$.[1] We refer to such a tree as the *factoring tree* $\mathcal{T}$ associated with the given polynomial $f(x)$. We describe next how $\mathcal{T}$ is constructed (and, hence, make more explicit its definition).

Consider $f(x)$ and $w(x)$ of the form (1) with $d_r = k_r = n$. We use

$$\left(\sum_{i=0}^{r} x^{k_i}\right)\left(\sum_{j=0}^{r} x^{n-k_j}\right) = w(x)\widetilde{w}(x) = f(x)\tilde{f}(x) = \left(\sum_{i=0}^{r} x^{d_i}\right)\left(\sum_{j=0}^{r} x^{n-d_j}\right) \tag{3}$$

to determine possible values of A and B for the node (A, B). Observe that since $f(x)$ is a fixed given polynomial, the right-hand side of (3) is known. We are seeking then values for k_j which, when substituted into the left-hand side of (3), produce the known exponents on the right-hand side of (3).

Initially, we introduce some simplifications. An exponent e appears on the right or left of (3) if and only if the exponent $2n-e$ does as well. Also, the exponent n occurs on both sides of (3) with multiplicity $r+1$. Thus, we simply need to equate the exponents $< n$ together with their multiplicities on each side of (3). If we can find k_j for which these exponents agree, then (3) will be satisfied.

At the outset the coefficients $k_0 = 0$ and $k_r = n$ are known. Label the first node of the factoring tree $([0],[0])$ (since $k_0 = 0$, the least exponent appearing in $w(x)$ is 0; since $k_r = n$, the least exponent appearing in $\widetilde{w}(x)$ is $n - k_r = 0$). This node then contains the known information about k_0 and k_r. In general, the nodes will be of the form (A, B) where $A = [k_0, k_1, \ldots, k_i]$ and $B = [n-k_r, n-k_{r-1}, \ldots, n-k_j]$ for some $i \geq 0$ and $j \geq 0$ with $i < j$. Here, the quantities $k_0, k_1, \ldots, k_i$ and $k_r, k_{r-1}, \ldots, k_j$ represent numbers being considered for the corresponding exponents in $w(x)$. Observe that if $i = j-1$, then $w(x)$ is completely determined. To create the next nodes in the tree, put $k_0, k_1, \ldots, k_i$ and $k_r, k_{r-1}, \ldots, k_j$ into (3), expand the left-hand side, and cancel any terms where the exponents agree with the right-hand side. For example, beginning with $([0],[0])$ (*i.e.*, $k_0 = 0$ and $k_r = n$), there are only four exponents on the left-hand side (including multiplicity) that do not involve some unknown, so those four exponents are what we cancel from both sides of the equation. We look at the smallest positive exponent that still occurs on the right; call it α. The minimal exponent remaining on the left-hand side must equal α. This minimal exponent will be of the form $k_u + n - k_v$ where at least one of k_u and k_v is an unknown. In other words, either $i < u < j$ or $i < v < j$. In the case that $i < u < j$, the minimum value of $k_u + n - k_v$ is obtained by taking $v = r$ and $u = i+1$. In the case that $i < v < j$, we obtain the minimum value of $k_u + n - k_v$ by taking $u = 0$ and $v = j-1$. Thus, we branch off into two new possibilities, each of which determines exactly one more exponent in $w(x)$ than the original node. In the first case $k_{i+1} = \alpha$ and in the second case $n - k_{j-1} = \alpha$. Thus, the two new nodes are

$$([k_0, k_1, \ldots, k_i, \alpha], [n-k_r, n-k_{r-1}, \ldots, n-k_j])$$

and

$$([k_0, k_1, \ldots, k_i], [n-k_r, n-k_{r-1}, \ldots, n-k_j, \alpha]).$$

[1]Since a 0,1-polynomial is uniquely determined by the exponents that appear in the polynomial, it is convenient to express a 0,1-polynomial as a list of these exponents.

After $r-1$ branchings from the initial node $([0],[0])$, $w(x)$ will necessarily be associated with the values of k_j determined by one of the endnodes from the last branching of the factoring tree $\mathcal{T}$. There are no more than 2^{r-1} endnodes corresponding to no more than 2^{r-1} possible values of $w(x)$. Our construction of the factoring tree can be summarized as follows.

Algorithm T *(Construct Factoring Tree)*: Given $f(x) = \sum_{j=0}^{r} x^{d_j} \in S$ with $0 = d_0 < d_1 < d_2 < \cdots < d_{r-1} < d_r = n$, construct the factoring tree associated with $f(x)$.

Step T1. *Initialize.* Compute the ordered list E of $r(r+1)/2$ increasing exponents $n-d_j+d_i$ where $0 \le i < j \le r$ (appearing on the right-hand side of (3)). Delete the number 0 from this list (the exponent $n - d_r + d_0$).

Step T2. *Set first node.* Set the value of the first node to be $([0],[0])$. Set an exponent list $E([0],[0])$ associated with the node $([0],[0])$ to be E (defined in Step T1). Set $i = 0$ and $j = r$. Set the level ℓ to be 1.

Step T3. *Start to develop nodes at level $\ell + 1$.* Begin a loop through the nodes at level ℓ. For each $N = ([k_0, k_1, \ldots, k_i], [n - k_r, n - k_{r-1}, \ldots, n - k_j])$ at the level ℓ, set the minimal element of $E(N)$ to be α and delete it from the list to form a new list $E'(N)$ (remove only one copy of α from $E(N)$ if more than one occurs). Set $E''(N) = E'(N)$.

Step T4. *Consider $k_{i+1} = \alpha$ at node N.* Compute $\alpha + n - k_t$ for $j \le t \le r$. For each such t in turn, revise $E'(N)$ by deleting one copy of $\alpha + n - k_t$ from list $E'(N)$ if it exists. If $\alpha + n - k_t \notin E'(N)$ for some t, then go to Step T5. Otherwise, add the node $N' = ([k_0, k_1, \ldots, k_i, \alpha], [n - k_r, n - k_{r-1}, \ldots, n - k_j])$ (so that $k_{i+1} = \alpha$) to level $\ell + 1$, and set $E(N') = E'(N)$.

Step T5. *Consider $n - k_{j-1} = \alpha$ at node N.* Compute $\alpha + k_t$ for $0 \le t \le i$. For each such t in turn, revise $E''(N)$ by deleting one copy of $\alpha + k_t$ from list $E''(N)$ if it exists. If $\alpha + k_t \notin E''(N)$ for some t, then go to Step T6. Otherwise, add the node $N'' = ([k_0, k_1, \ldots, k_i], [n - k_r, n - k_{r-1}, \ldots, n - k_j, \alpha])$ (so that $n - k_{j-1} = \alpha$) to level $\ell + 1$ and set $E(N'') = E''(N)$.

Step T6. *Check if level is finished.* If there are more nodes to consider at the current level, then continue with Step T3. Otherwise, check if the current level is $r - 1$. If so, the nodes on level r have just been created and the factoring tree is complete; stop. If not, check if there exist any nodes at the next level. If there are no nodes on level $\ell + 1$ the factoring tree is complete with no endnodes at level r; stop. Otherwise, increment the level ℓ by 1 and begin the loop at Step T3 for the new level.

Before continuing, we discuss the running time of Algorithm T. For each i and j with $0 \le i < j \le r$, computing $n - d_j + d_i$ takes $\ll \log n$ bit operations so the elements of E can be computed in $\ll r^2 \log n$ bit operations. Ensuring that the elements in E are ordered increases the time estimate by a factor of $\ll \log r$. Hence the total time in Step T1 is $\ll r^2 \log r \log n$. A bound on the number of bits occupied by the list E is $\ll r^2 \log n$. It follows that Step T2 requires at most $\ll r^2 \log n$ bit operations. The factoring tree contains one node at the first level, and at each subsequent level the number of nodes at most doubles

(from Steps T4 and T5) so that the number of nodes at level ℓ does not exceed $2^{\ell-1}$. Suppose we are at level $\ell \geq 1$. In Steps T3-T5, each new node N created in level $\ell+1$ is assigned an increasing exponent list consisting of less than $|E| \ll r^2$ elements. Each element in each exponent list is no larger than n. Because the exponent list is kept in increasing order, the least element in a given exponent list is always the first element. Since there are $\leq 2^{\ell-1}$ nodes at level ℓ, we deduce that Step T3 takes $\ll 2^{\ell} \log n$ bit operations performed at level ℓ. Computing the $r-j+1$ numbers $\alpha+n-k_t$ in Step T4 takes at most $\ll r \log n$ bit operations. Also, exploiting the fact that the elements in $E'(N)$ are ordered, the number of bit operations for determining whether a given $\alpha+n-k_t$ is in $E'(N)$ (and deleting it if it is) is $\ll \log |E'(N)| \log n \ll \log r \log n$. Forming N' and $E(N')$ then requires at most $\ll r \log r \log n$ bit operations. Since this is done for each node N at level ℓ, the amount of time spent on level ℓ in Step T4 is $\ll 2^{\ell} r \log r \log n$. Similarly, Step T5 requires running time $\ll 2^{\ell} r \log r \log n$. Step T6 only requires $\ll \log r \ll \log n$ bit operations. We deduce that the running time for Algorithm T is

$$\ll \sum_{\ell=1}^{r-1} 2^{\ell} r \log r \log n \ll 2^{r} r \log r \log n.$$

4 A Proof of the Theorem

We are now ready to describe an algorithm for determining if the non-reciprocal part of a given polynomial is irreducible.

Algorithm NR *(Determine Irreducibility of the Non-Reciprocal Part):* Given the polynomial $f(x) = \sum_{j=0}^{r} x^{d_j} \in S$ with $d_0 = 0 < d_1 < d_2 < \cdots < d_{r-1} < d_r = n$, determine if the non-reciprocal part of $f(x)$ is irreducible.

Step NR1. *Check if $f(x)$ is reciprocal.* Check if $d_i = n - d_{r-i}$ for every integer $i \in [0, r/2]$. If so, stop and output the non-reciprocal part of $f(x)$ is 1 and therefore not irreducible.

Step NR2. *Construct the factoring tree.* Construct the factoring tree $\mathcal{T}$ (using Algorithm T) associated with $f(x)$ having at most $2^r - 1$ total nodes and at most 2^{r-1} endnodes.

Step NR3. *Check polynomials associated with the endnodes.* If there are no endnodes on level r of the factoring tree or if each endnode on level r corresponds to either $w(x) = f(x)$ or $w(x) = \tilde{f}(x)$, then the algorithm terminates with a message that the non-reciprocal part of $f(x)$ is irreducible. Otherwise, select an endnode on level r that corresponds to a polynomial $w(x)$ such that $w(x) \neq f(x)$ and $w(x) \neq \tilde{f}(x)$ and terminate the algorithm by reporting that the non-reciprocal part of $f(x)$ is reducible and that $w(x)$ is a polynomial with the property that "$\gcd(f(x), w(x))$" is a factor of $f(x)$.

We explain the correctness of the algorithm. If $d_i = n - d_{r-i}$ for every integer $i \in [0, r/2]$, then the input polynomial $f(x)$ is reciprocal. It follows from Lemma 2 that the non-reciprocal part of $f(x)$ is 1 and not irreducible. If $d_i \neq n - d_{r-i}$ for some integer $i \in [0, r/2]$, then $f(x)$ is not reciprocal and, hence, $f(x)$ has a non-reciprocal factor of degree at least 1. By

the construction of the factoring tree $\mathcal{T}$, if $w(x) \in S_r$ and (2) holds, then the exponents k_j of $w(x)$ are determined by one of the endnodes at level r of $\mathcal{T}$. By Lemma 1, if some $w(x)$ formed from such an endnode satisfies $w(x) \neq f(x)$ and $w(x) \neq \tilde{f}(x)$, then the non-reciprocal part of $f(x)$ is reducible. Hence, in this case, the non-reciprocal part of $f(x)$ is not irreducible. If no $w(x)$ formed from the endnodes at level r of $\mathcal{T}$ satisfies $w(x) \neq f(x)$ and $w(x) \neq \tilde{f}(x)$, then Lemma 2 implies that the non-reciprocal part of $f(x)$ is not reducible and, hence, is irreducible. This justifies the algorithm.

To complete the proof, the running time of the algorithm will be shown to be $\ll 2^r r \log r \log n$. In Step NR1, we check for each $i \in [0, r/2]$ whether $d_i = n - d_{r-i}$. Each such check requires at most $\ll \log n$ bit operations, so that the time spent in Step NR1 is $\ll r \log n$. As indicated at the end of the previous section, the running time for Step NR2 is $\ll 2^r r \log r \log n$. There are at most 2^{r-1} nodes at level r of the factoring tree. For each such node, constructing $w(x) = \sum_{j=0}^{r} x^{k_j}$ takes on the order of $\ll r \log n$ bit operations. For each such $w(x)$, the checks to see if $w(x) \neq f(x)$ and $w(x) \neq \tilde{f}(x)$ can be completed by comparing sorted exponent lists; these comparisons can be performed in $\ll r \log r \log n$ bit operations. Hence, Step NR3 takes at most $\ll 2^r r \log r \log n$ bit operations. Theorem 1 follows.

References

[1] M. Filaseta, *On the factorization of polynomials with small Euclidean norm*, Number theory in progress, Vol. 1 (Zakopane-Kościelisko, 1997), de Gruyter, Berlin, 1999, 143–163.