# On Computation of Polynomial Modular Reduction

Huapeng Wu*

June 9, 2000

**Abstract**

In this paper, we consider the problem of efficient computation of polynomial modular reduction: $A(x) \bmod f(x)$, where $f(x)$ is a monic polynomial of degree $n$ and $A(x)$ is a polynomial of degree not greater than $n+t-1, t \geqslant 1$, both $f(x)$ and $A(x)$ are defined over a commutative ring $R$ with identity. For given $f(x)$ and the degree $n+t-1$ of $A(x)$, we present an algorithm to compute this problem in $t(w-1)$ addition operations in $R$ and the same number of multiplication operations in $R$, where $w$ is the Hamming weight of $f(x)$. Applications of the proposed algorithm to finite field arithmetic are also discussed.

**Key Word:**

Polynomial arithmetic, modular operation, finite field arithmetic, complexity.

# 1. INTRODUCTION

The recent advances in public key cryptography, especially elliptic curve cryptography, have rekindled the research in polynomial arithmetic, which is required in many finite field operations. One example is finite field multiplication. Let $f(x)$ be an irreducible polynomial over GF($q$) of degree $n$. Then $\{1, x, \ldots, x^{n-1}\}$ forms a standard basis in GF($q^n$) over GF($q$). Every element in

*H. Wu is with the Centre for Applied Cryptographic Research, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Canada N2L 3G1. E-mail:h3wu@cacr.math.uwaterloo.ca .

GF($q^n$) can be represented with a polynomial over GF($q$) of degree not greater than $n-1$. Then a multiplication operation in GF($q^n$) can be realized in two steps: First we perform polynomial multiplication and obtain a product polynomial of degree not greater than $2n-2$; Then the degree of this product polynomial is reduced to the proper degree range by applying to it polynomial modular operation.

We consider here the problem of efficient computation of polynomial modular reduction: $A(x) \bmod f(x)$, where $f(x)$ is a monic polynomial of degree $n$ and $A(x)$ is a polynomial of degree not greater than $n+t-1$, the coefficients of both $f(x)$ and $A(x)$ are defined over a commutative ring $R$ with identity. One way to obtain polynomial modular reduction is to use the well known polynomial division algorithm ([2], pp 402).[1] It has a complexity of $O\big(n(n+t)\big)$ operations in $R$. Let $w$ denote the Hamming weight of $f(x)$. Then a complexity upper bound of $(w-1)(n-1)$ bit operations is also known for the second step of a standard basis finite field multiplication as explained in the last paragraph, if $q=2$ [1, 3]. To our knowledge, however, there is no explicit algorithm available in the literature regarding to how efficiently computing polynomial modular reduction fitting the above complexity bound.

In this paper, for given $f(x)$ of degree $n$ and the degree upper bound $n+t-1$ of $A(x)$, we present an algorithm to compute this problem in $t(w-1)$ addition operations in $R$ and the same number of multiplication operations in $R$, where $w$ is the Hamming weight of $f(x)$. Applications of this algorithm to finite field arithmetic are also discussed.

The organization of this paper is as follows: We present the algorithm in §2.1 and illustrate how it works with an example in §2.2. The complexity issue is discussed in §2.3. Modifications of the algorithm to suit for polynomials over finite field of characteristic two are made in Section 3. The complexities of finite field multiplication and squaring operation are respectively discussed in §3.1 and §3.2.

[1] In this algorithm, polynomials are required to be defined over a field. However, if the divisor polynomial is a monic polynomial, then this method also applies to polynomials over a commutative ring with identity.

## 2. Efficient Algorithm for Polynomial Modular Reduction

### 2.1. Algorithm

Let $f(x)$ be a monic polynomial of Hamming weight $w$ over a commutative ring $R$ with identity and be given by

$$f(x) = x^n + f_{e_{w-2}} x^{e_{w-2}} + \cdots + f_{e_1} x^{e_1} + f_{e_0} x^{e_0},$$

where $n > e_{w-2} > e_{w-3} > \cdots > e_1 > e_0 \geqslant 0$. Let the polynomial $A(x) = \sum_{i=0}^{m} a_i x^i,\ a_i \in R$, whose degree is to be reduced have its degree bounded by $m \leqslant n + t - 1$. Then polynomial modular operation is given by

$$B(x) = A(x) \bmod f(x). \tag{1}$$

If $f(x)$ is a monomial, then $B(x)$ is simply a sum of those terms of $A(x)$ whose degree is not higher than $n-1$. If $f(x)$ has a more complex form, then we have the following algorithm to compute the polynomial modular reduction operation (1).

**Algorithm 1** Reduction modulo a polynomial

Input: $f(x), A(x)$.

Output: $B(x) = A(x) \bmod f(x)$, where $\deg B < \deg f$.

Part 1. Precomputation

Input: $f(x)$ and the upper bound $n+t-1$ of $\deg(A)$.

Output: Prepared coefficient lists $l_0, l_1, \ldots, l_{n+t-1}$.

1. Initialization of coefficient lists:

   $$x^j :\ l_j = \langle a_j \rangle,\ j = 0, 1, \ldots, n+t-1.$$

2. Compute the prepared coefficient lists:

   For $i = t-1$ To $0$, Step $-1$

   For $j = 0$ To $w-2$

   Append the pair $(a_{n+i}, f_{e_j})$ as one element to the coefficient list $l_{i+e_j}$ .

Part 2. Main Program

Input: The coefficients of $A(x)$, and the prepared coefficient lists $l_0, l_1, \ldots, l_{n+t-1}$.
Output: The coefficients of $B(x)$: $b_0, b_1, \ldots, b_{n-1}$.

For $i = t + e_{w-2} - 1$ To $0$, Step $-1$

(i). Compute the product of the two terms of a pair for all the pairs in $l_i$;

(ii). $a_i \Leftarrow$ the sum of all the elements in $l_i$;

Output as results: $b_j = a_j, j = 0, 1, \ldots, n-1$.

First, for each term $x^i$ of $A(x)$, $i = 0, 1, \ldots, n+t-1$, a *coefficient list* (CL) $l_i$ is introduced, which initially has the coefficient $a_i$. Note that in the precomputation part, the coefficients $a_i$ of the polynomial $A(x)$ are unknown and they are treated as variables. We extend the terms $x^{n+i}, i = 0, 1, \ldots, t-1$, as follows:

$$x^{n+i} \bmod f(x) = f_{e_{w-2}} x^{e_{w-2}+i} + f_{e_{w-3}} x^{e_{w-3}+i} + \cdots + f_{e_0} x^{e_0+i}.$$

For each term $f_{e_j} x^{e_j+i}$ on the right-hand side of the above expression, we append the pair $(f_{e_j}, a_{n+i})$ to $l_{e_j+i}$. Then after precomputation each CL $l_i$ contains one element $a_i$ and possibly a few other elements of the form of a pair $(f_{i1}, a_{i2})$, and the list is now referred to as a *prepared* CL. Part two of Algorithm 1 is to assign value to variable $a_i$ and computation is performed on the CLs. In the following we first show the correctness of the algorithm with an example. Then complexity of the algorithm is analyzed.

## 2.2. An Example

Let the monic polynomial $f(x)$ be given as $f(x) = x^5 - 4x^3 - 2$, and $R$ be the integer ring $\mathbb{Z}$. Let the polynomial whose degree is to be reduced by modular operation is

$$A(x) = 6x^8 + 2x^7 + x^6 + 6x^5 + 3x^3 + 5. \tag{2}$$

In following we proceed with Algorithm 1 when it takes inputs of $f(x)$ and $A(x)$ given above.

1. Precomputation: preparing coefficient lists.

(i). The coefficient lists are initialized:

$$\begin{array}{ll} x^0: & l_0 = \langle a_0 \rangle \\ x^1: & l_1 = \langle a_1 \rangle \\ x^2: & l_2 = \langle a_2 \rangle \\ x^3: & l_3 = \langle a_3 \rangle \\ x^4: & l_4 = \langle a_4 \rangle \\ x^5: & l_5 = \langle a_5 \rangle \\ x^6: & l_6 = \langle a_6 \rangle \\ x^7: & l_7 = \langle a_7 \rangle \\ x^8: & l_8 = \langle a_8 \rangle \end{array} \tag{3}$$

(ii). The coefficient lists (3) can be updated as

$$\begin{array}{llllll} x^0: & l_0 = \langle a_0 \rangle & \xrightarrow{\text{(4d)}} & l_0 = \langle a_0, (2, a_5) \rangle & & \\ x^1: & l_1 = \langle a_1 \rangle & \xrightarrow{\text{(4c)}} & l_1 = \langle a_1, (2, a_6) \rangle & & \\ x^2: & l_2 = \langle a_2 \rangle & \xrightarrow{\text{(4b)}} & l_2 = \langle a_2, (2, a_7) \rangle & & \\ x^3: & l_3 = \langle a_3 \rangle & \xrightarrow{\text{(4a)}} & l_3 = \langle a_3, (2, a_8) \rangle & \xrightarrow{\text{(4d)}} & l_3 = \langle a_3, (2, a_8), (4, a_5) \rangle \\ x^4: & l_4 = \langle a_4 \rangle & & & \xrightarrow{\text{(4c)}} & l_4 = \langle a_4, (4, a_6) \rangle \\ x^5: & l_5 = \langle a_5 \rangle & & & \xrightarrow{\text{(4b)}} & l_5 = \langle a_5, (4, a_7) \rangle \\ x^6: & l_6 = \langle a_6 \rangle & & & \xrightarrow{\text{(4a)}} & l_6 = \langle a_6, (4, a_8) \rangle \\ x^7: & l_7 = \langle a_7 \rangle & & & & \\ x^8: & l_8 = \langle a_8 \rangle & & & & \end{array}$$

This step can be explained as follows:

For the terms whose degree is equal to or higher than that of $f(x)$, and equal to or lower than the bound $n + t - 1 = 8$, we extend them using the expressions $x^{5+i} = 4x^{3+i} + 2x^i,\ i = t-1, t-2, \ldots, 0$, or:

$$x^8 = 4x^6 + 2x^3 \tag{4a}$$

$$x^7 = 4x^5 + 2x^2 \tag{4b}$$

$$x^6 = 4x^4 + 2x^1 \tag{4c}$$

$$x^5 = 4x^3 + 2x^0 \tag{4d}$$

By using (4a-4d), the coefficient lists (3) can be updated based on the following argument. For the first expression (4a), as an example, after (4a) has been applied to $A(x)$ to reduce its degree, the coefficient of the term $x^8$ should be added to these of the terms $x^6$ and $x^3$, or added to the coefficient lists $l_6$ and $l_3$. Since the term $x^8$ initially has the coefficient $a_8$, after using (4a) for $A(x)$, the coefficient of $x^6$ should be updated as $(a_6+4\times a_8)$, and the coefficient of $x^3$ should be updated as $(a_3+2\times a_8)$. Consequently, we have the CLs for $x^6$ and $x^3$ updated as $\langle a_6,(f_3,a_8)\rangle$ and $\langle a_3,(f_0,a_8)\rangle$, respectively. The rest undates can be done based on the expressions (4b-4d) with similar argument.

(iii). The precomputed or the *prepared* coefficient list can be given as

$$\begin{array}{ll}
x^0: & l_0=\langle a_0,(2,a_5)\rangle \\
x^1: & l_1=\langle a_1,(2,a_6)\rangle \\
x^2: & l_2=\langle a_2,(2,a_7)\rangle \\
x^3: & l_3=\langle a_3,(2,a_8),(4,a_5)\rangle \\
x^4: & l_4=\langle a_4,(4,a_6)\rangle \\
x^5: & l_5=\langle a_5,(4,a_7)\rangle \\
x^6: & l_6=\langle a_6,(4,a_8)\rangle \\
x^7: & l_7=\langle a_7\rangle \\
x^8: & l_8=\langle a_8\rangle
\end{array} \tag{5}$$

2. Main program

Input: The prepared coefficient lists and the coefficients of $A(x)$:
$a_8=6,a_7=2,a_6=1,a_5=6,a_3=3,a_0=5,a_i=0,i=1,2,4.$

Output: Coefficients of $B(x)$: $b_0,b_1,b_2,b_3,b_4$.

Compute:

$$\begin{array}{lll}
a_6=a_6+4a_8 & \Rightarrow & a_6=25 \\
a_5=a_5+4a_7 & \Rightarrow & a_5=14 \\
a_4=a_4+4a_6 & \Rightarrow & a_4=100 \\
a_3=a_3+2a_8+4a_5 & \Rightarrow & a_3=71 \\
a_2=a_2+2a_7 & \Rightarrow & a_2=4 \\
a_1=a_1+2a_6 & \Rightarrow & a_1=50 \\
a_0=a_0+2a_5 & \Rightarrow & a_0=33
\end{array}$$

The output is $b_i=a_i,i=0,1,2,3,4$. So we have

$$6x^8+2x^7+x^6+6x^5+3x^3+5 \bmod x^5-4x^3-2=100x^4+71x^3+4x^2+50x+33.$$

Clearly, the cost of Main program of Algorithm 1 (part 2) is $8$ addition operations in $\mathbb{Z}$ and $8$ constant multiplication operations (The multiply is a constant) in $\mathbb{Z}$.

It can be seen that Algorithm 1 it does not require $f(x)$ to be completely fixed. In fact, the values of the coefficients can be variable as long as the degree and the distribution of Hamming weight of $f(x)$ are fixed. In the above example, the precomputation step can still be performed even if we do not know the values of $f_3$ and $f_0$, as long as we know that $f_3$ and $f_0$ are the only coefficients that can be nonzero besides $f_n = 1$. In that case, $8$ multiplication operations (instead of $8$ constant multiplication operations) are required.

## 2.3. Complexity

It can be seen that the complexity of Algorithm 1 (Part 2) depends on the size of $t + n$ coefficient lists. During the precomputation, each coefficient list $l_i$ is first initialized to have one term $a_i$. Then, in the second step in precomputation, the $t + n$ coefficient lists are expanded by $t(w - 1)$ terms. These $t(w - 1)$ terms are in the form of a pair. So the total number of terms in the $t + n$ prepared CLs is $t + n + t(w - 1) = wt + n$.

The complexity of Algorithm 1 (Part 2) is decided by the steps (i) and (ii), where the sum of all the terms in a CL is obtained. Note that in the process of summing up, the product is used if a term is in the form of a pair. Since there are $n + t$ non-empty coefficient lists and they contain $wt + n$ terms in total (of which $(w - 1)t$ terms are pairs), we conclude that $wt + n - (n + t) = (w - 1)t$ addition operations in $R$ and the same number of multiplication operations in $R$ are required for Algorithm 1 (Part 2).

Since there are total $wt + n$ elements of which $t(w - 1)$ are pairs, the amount for the required memory for storing the prepared CLs is $wt+n+t(w-1) = 2wt+n-t$ units, with each unit storing one element in $R$. Note that only $t + e_{w-2}$ CLs are used in Part 2 of Algorithm 1. The required memory amount can be reduced by storing only those CLs that are used in the Main program. Then the necessary memory should have $2wt + n - t - (n - e_{w-2}) = 2wt + e_{w-2} - t$ units. To further save memory we may store in CLs the indices of the coefficients instead of their values.

# 3. Application in Finite Field Arithmetic

## 3.1. Polynomial basis multiplication

**Algorithm** Let the finite field GF($q^n$) be defined by the irreducible polynomial $f(x)$, thus $\{1, x, x^2, \ldots, x^{n-1}\}$ is a basis in GF($q^n$) over GF($q$). Let $A(x)$ and $B(x)$ be two field elements in GF($q^n$). Then a finite field multiplication $C(x) = A(x)B(x)$ can be realized in the following two steps:

1.

$$A(x)B(x) = S(x) = \sum_{i=0}^{2n-2} s_i x^i, \text{ where } s_i = \begin{cases} \sum_{j=0}^{i} a_j b_{i-j}, & 0 \leqslant i \leqslant n-1, \\ \sum_{j=i-n+1}^{n-1} a_j b_{i-j}, & m \leqslant i \leqslant 2n-2. \end{cases}$$

2.

$$C(x) = S(x) \bmod f(x). \tag{6}$$

**Complexity** Clear, Algorithm 1 can be directly used to compute the expression (6) with $t = n-1$. It has a complexity of $(w-1)(n-1)$ addition operations in GF($q$) and $(w-1)(n-1)$ constant multiplication operations in GF($q$), where $w$ is the Hamming weight of the monic irreducible polynomial $f(x)$. If $f(x)$ is chosen to have a low Hamming weight, *i.e.*, binomial and trinomial, then only $O(n)$ operations in the ground field are required for reduction modulo a polynomial.

**Reduction modulo a polynomial over GF$(2)$** In the case that $q = 2$, since a constant in GF$(2)$ is either $0$ or $1$, the constant multiplication operations can be saved. Thus the step of reduction modulo the irreducible polynomial requires only $(w-1)(n-1)$ addition operations in GF$(2)$. A version of Algorithm 1 over GF$(2)$ is given below.

**Algorithm 2** Polynomial modular reduction over GF$(2)$
Input: $f(x), A(x)$.
Output: $B(x) = A(x) \bmod f(x)$, where $\deg B < \deg f$.

Part 1. Precomputation
Input: $f(x)$ and the degree upper bound $n+t-1$ of $A(x)$.
Output: Prepared coefficient lists $l_0, l_1, \ldots, l_{n+t-1}$.

1. Initialization of coefficient lists:

$$x^j: \quad l_j = \langle a_j \rangle, \quad j = 0, 1, \ldots, n+t-1.$$

2. Compute the prepared CLs:

For $i = t-1$ To $0$, Step $-1$

For $j = 0$ To $w-2$

Append $a_{n+i}$ to $l_{i+e_j}$ .

Part 2. Main Program
Input: The coefficients of $A(x)$, and the prepared coefficient lists $l_0, l_1, \ldots, l_{n+t-1}$.
Output: The coefficients of $B(x)$: $b_0, b_1, \ldots, b_{n-1}$.

For $i = t + e_{w-2} - 1$ To $0$, Step $-1$

$a_i \Leftarrow$ the sum of all the terms in $l_i$;

Output as results: $b_j = a_j, j = 0, 1, \ldots, n-1$.

It can be seen that Algorithm 2 can be used for computing the expression (6) with $t = n - 1$.

### 3.2. Polynomial basis Squaring in GF$(2^n)$

**Algorithm** Assume that $f(x) = x^n + \sum_{i=0}^{w-2} x^{e_i}$ is an irreducible polynomial over GF$(2)$. Let $A(x)$ be a field element in GF$(2^n)$ and be represented in the polynomial basis expanded from a root of $f(x)$. Then squaring operation $C(x) = A^2(x) \bmod f(x)$ in GF$(2^n)$ is given by

$$\sum_{i=0}^{n-1} a_i x^{2i} = \sum_{i=0}^{2n-2} s_i x^i, \text{ where } s_i = \begin{cases} a_{\frac{i}{2}}, & i = 0, 2, \ldots, 2n-2; \\ 0, & i = 1, 3, \ldots, 2n-3. \end{cases}$$

In this case of application of Algorithm 1, since we have additional information on the polynomial that is to be reduced ($s_i = 0$ for $i$ odd), the precomputation part is done in a slightly different way from that in Algorithm 2.

**Algorithm 3** Polynomial basis squaring in GF$(2^n)$

Input: $f(x), S(x) = A^2(x)$.

Output: $B(x) = S(x) \bmod f(x)$, where $\deg B < \deg f$.

Part 1. Precomputation

Input: $f(x)$.

Output: Prepared coefficient lists $l_0, l_1, \ldots, l_{2n-2}$.

1. Initialization of coefficient lists:

   $x^j: \quad l_j = \langle a_j \rangle, \quad j = 0, 2, \ldots, 2n-2.$

   $x^j: \quad l_j = \emptyset,$ otherwise.

2. Compute the prepared CLs:

   For $i = 2n-2$ To $n$, Step $-2$

   For $j = 0$ To $w-2$

   Append $a_i$ to $l_{i-n+e_j}$ .

$$\text{Let } h = \begin{cases} \max_{0 \leqslant i \leqslant w-2}\{n-2+e_i | n+e_i \text{ is odd.}\} & \text{if } e_i > 0, \\ n-1 & \text{otherwise.} \end{cases} \tag{7}$$

   If $(h > n)$ Then

   For $i = h$ To $n$, Step $-2$

For $j = 0$ To $w - 2$

Append $a_i$ to $l_{i-n+e_j}$ .

Part 2. Main Program

Input: The coefficients of $S(x)$, and the prepared coefficient lists $l_0, l_1, \ldots, l_{n+e_{w-2}-2}$.

Output: The coefficients of $B(x)$: $b_0, b_1, \ldots, b_{n-1}$.

For $i = n + e_{w-2} - 2$ To $0$, Step $-1$

$a_i \Leftarrow$ the sum of all the terms in $l_i$;

Output as results: $b_j = a_j, j = 0, 1, \ldots, n-1$.

**Complexity** Let $N_1$ denote the number of terms appended to the $2n-1$ CLs in Step 2 of the precomputation part in Algorithm 3. Then we have

$$N_1 = \begin{cases} \left\lceil \frac{n-1}{2} \right\rceil (w-1) + \left\lceil \frac{h-n+1}{2} \right\rceil (w-1) & \text{if } h \geq n, \\ \left\lceil \frac{n-1}{2} \right\rceil (w-1) & \text{otherwise,} \end{cases}$$

Let the number of bit additions required in Part 2 in Algorithm 3 be denoted by $N$. Clearly, $N \leqslant N_1$. However, the CL $l_i$ with $i$ being odd was initially empty and thus in part two of the algorithm one bit operation can be saved in summing up the elements in $l_i$ if it becomes non-empty after step 2 of part 1. Let $N = N_1 - \phi$. Then $\phi$ is the number of the different odd values of $i - n + e_j$ for $i = 2n-2, 2n-4, \ldots, 2 \times \left\lceil \frac{n}{2} \right\rceil$ and $j = 0, 1, \ldots, w-2$. If $f(x)$ is irreducible over GF(2), then at least one of $n$ and $e_j, j = 0, 1, \ldots, w-2$ is an odd number. We thus have $\phi \geqslant \left\lceil \frac{n-1}{2} \right\rceil$. Therefore, it follows

$$\begin{aligned} N &= N_1 - \phi \\ &\leqslant N_1 - \left\lceil \frac{n-1}{2} \right\rceil \\ &= \begin{cases} \left\lceil \frac{n-1}{2} \right\rceil (w-2) + \left\lceil \frac{h-n+1}{2} \right\rceil (w-1) & \text{if } h \geqslant n \\ \left\lceil \frac{n-1}{2} \right\rceil (w-2) & \text{otherwise,} \end{cases} \end{aligned} \tag{8}$$

where $h$ is defined in (7) in Algorithm 3.

Consider $f(x)$ to be an irreducible trinomial $x^n + x^k + 1$. We try to use the bound (8) to decide the complexity for this case. From (7), we have

$$h = \begin{cases} n+k-2 & \text{if } n+k \text{ is odd,} \\ \\ n-1 & \text{otherwise.} \end{cases}$$

Then from (8) and note $w = 3$, it gives the bound below:

$$N = \begin{cases} \left\lceil \frac{n-1}{2} \right\rceil + 2\left\lceil \frac{k-1}{2} \right\rceil & \text{if } n+k \text{ is odd,} \\ \frac{n-1}{2} & \text{if } n+k \text{ is even.} \end{cases}$$

# References


[1] V. B. Afanasyev, C. Gehrmann and B. Smeets, Fast message authentication using efficient polynomial evaluation, Fast Software Encryption Workshop (E. Biham, Ed.), Lecture Notes in Computer Science, Springer-Verlag, New York, 1267 (1997) pp. 190-204.

[2] D. E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, Addison-Wesley Publishing Company, Reading, MA (1981).

[3] H. Wu, *Efficient Computations in Finite Fields with Cryptographic Significance*, Ph.D Thesis, University of Waterloo, Waterloo,Ontario,Canada (1998).

## Key Words:

Polynomial arithmetic, modular operation, finite field arithmetic, complexity.

**Postal:**

Huapeng Wu

The Centre for Applied Cryptographic Research

Dept of Combinatorics and Optimization

University of Waterloo

Waterloo Ontario

Canada N2L 3G1

**Voice:**

519-888-4567 x3600

**Fax:**

519-725-5441

**Email:**

h3wu@cacr.math.uwaterloo.ca

**Affiliation of Author:**

The Centre for Applied Cryptographic Research

Dept of Combinatorics and Optimization

University of Waterloo

**Footnotes:**

1. In this algorithm, polynomials are required to be defined over a field. However, if the divisor polynomial is a monic polynomial, then this method also applies to polynomials over a commutative ring with identity.

Melissa Sullivan
Designs, Codes and Cryptography - Editorial Office
Kluwer Academic Publishers
101 Philip Drive
Norwell, MA 02061, U.S.A.