Journal of Computer Science & Computational Mathematics, *Volume 12, Issue 1, March 2022*
DOI: 10.20967/jcscm.2022.01.002

An Alternative Approach to Find the Greatest Common Divisor in Polynomials

A. N. M. Rezaul Karim

Department of Computer Science & Engineering, International Islamic University Chittagong, Bangladesh
Corresponding author email: zakianaser@yahoo.com

***Abstract*:** A new method for determining the greatest common divisor in polynomials has been developed. We reduced the amount of iterations in our algorithm, which now accounts for half of all Euclidean GCDP and EEAGCDP implementations. We can deduce that our proposed strategy is faster than the existing way since we eliminate some processing effort.

***Keywords*:** Greatest common divisor, extended Euclidean greatest common divisor for polynomials, Euclidean algorithm for polynomials.

1. Introduction

The problem considered in this paper is computation of the greatest common divisor of nonzero polynomials $M(x)$ and $P(x)$, where deg $M(x) <$ deg $P(x)$ in the Galois field $F(x) = GF(2^n)$, $n \geq 2$, which appears in many situations in cryptography. The process of computing the greatest common divisor is inseparable from computing the multiplicative inverse. In general, the extended Euclidean algorithm can be used to compute them.

To compute a greatest common divisor, we will use the more elegant theorem (presented in the paper [1]), which computes the multiplicative inverse in simple and straightforward steps.

2. Basic Definitions

2.1 Greatest Common Divisor (GCD)

Let $M(x)$ and $P(x)$ in $F(x)$, then the greatest common divisor of $M(x)$ and $P(x)$, denoted $gcd(M(x), P(x))$ is the polynomial of greatest degree in $F(x)$ which divides both $M(x)$ and $P(x)$ [2].

2.2 Irreducible polynomial

A polynomial $P(x) \in F(x)$, irreducible in $F(x)$ if $P(x)$ cannot be expressed as a product $A(x)B(x)$ of two polynomials $A(x)$ and $B(x)$ in $F(x)$ both of lower degree than the degree of $P(x)$ [3, 4].

2.3 Multiplicative inverse (MI)

The multiplicative inverse of $M(x)$ modulo $P(x)$ is $M^{-1}(x)$ such that

$$M(x)M^{-1}(x) = 1 \pmod{P(x)}. \tag{1}$$

We will denote it by $T[M(x)]$ [2].

3. Relationship between gcd and MI

Bezout identity can be stated as follows:

If $d(x) = gcd(M(x), P(x))$ for given polynomials $M(x)$ and $P(x)$ in $F(x)$, then there are two polynomials $A(x)$ and $B(x)$, (are not unique), such that

$$d(x) = A(x)M(x) + B(x)P(x). \tag{2}$$

If $d(x) = 1$, and since

$$B(x)P(x) = 0 \pmod{P(x)} \tag{3}$$

We get,

$$A(x)M(x) = 1 \pmod{P(x)} \tag{4}$$

Hence $A(x)$ is a multiplicative inverse of $M(x)$ modulo $P(x)$, this means a multiplicative inverse only exists when the gcd is 1.

Algorithm 01: Euclidean algorithm for polynomials is well known ([4–6])

Algorithm 02: Extended Euclidean algorithms for polynomials ([6, 7]) is:

Proposed Algorithm 03: We suggest the following algorithms. Here, two polynomials $a(x)$ and $b(x)$. (Figure 3)

Proposed Algorithm 04: Figure 4

Code for Algorithm 1:

```
\#include< bits/stdc++.h>

using namespace std;

int algorithm1(int ax, int bx)
{
  int r;
  while(bx!=0)
  {
     r=ax%bx;
     ax=bx;
     bx=r;
  }
  return ax;
}

int main()
{
  int a[100],b[100];
```

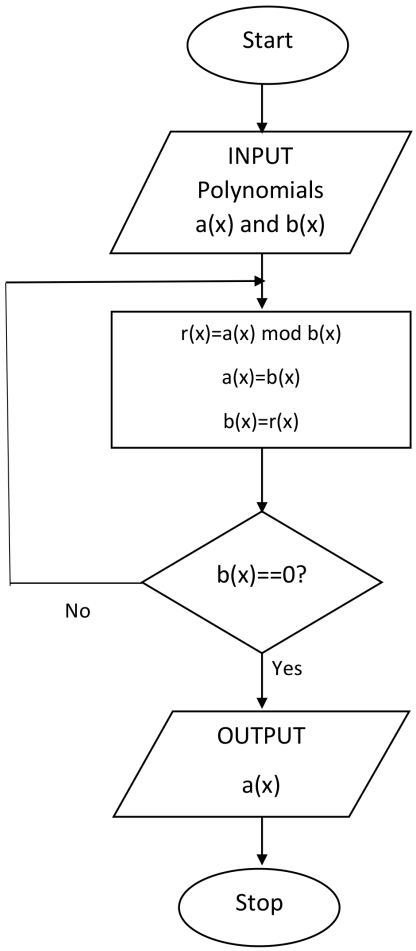

Figure 1. Graphical Representation of Algorithm 1

```
  int i, o_ax, o_bx,ra1;
  float x,x_bx, ax,bx;
printf("Enter the value of x for a(x)
  & b(x)\n");
scanf("%f", &x);
printf("Enter the order of the polynomial
 a(x)\n");
scanf("%d", &o_ax);
printf("Enter %d coefficients \n",
  o_ax + 1);
for (i = 0; i<= o_ax; i++)
scanf("%d", &a[i]);
printf("Enter the order of the polynomial
 b(x)\n");
scanf("%d", &o_bx);
printf("Enter %d coefficients \n",
 o_bx + 1);
   for (i = 0; i<= o_bx; i++)
   scanf("%d", &b[i]);
   ax = a[0];
   for (i = 1; i<= o_ax; i++)
   {
     ax = ax * x+ a[i];
   }
printf("The polynomial a(x) = %.0f \n", ax);
   bx = b[0];
   for (i = 1; i<= o_bx; i++)
   {
     bx = bx * x + b[i];
   }
printf("The polynomial b(x) = %.0f \n", bx);
printf("\nOutput of algorithm 1:\nThe
 greatest common divisor of ax & bx=%d\n",
 algorithm1(ax,bx));
   return 0;
}
output: Figure 5
```

Code for Algorithm 2:

```
\#include<bits/stdc++.h>

using namespace std;

void algorithm2(int ax,int bx)
{
   int s1x,s2x,t2x,t1x,qx,rx,sx,tx,a,b,dx;
   a=ax;
   b=bx;
   s2x=1;
   s1x=0;
   t2x=0;
```

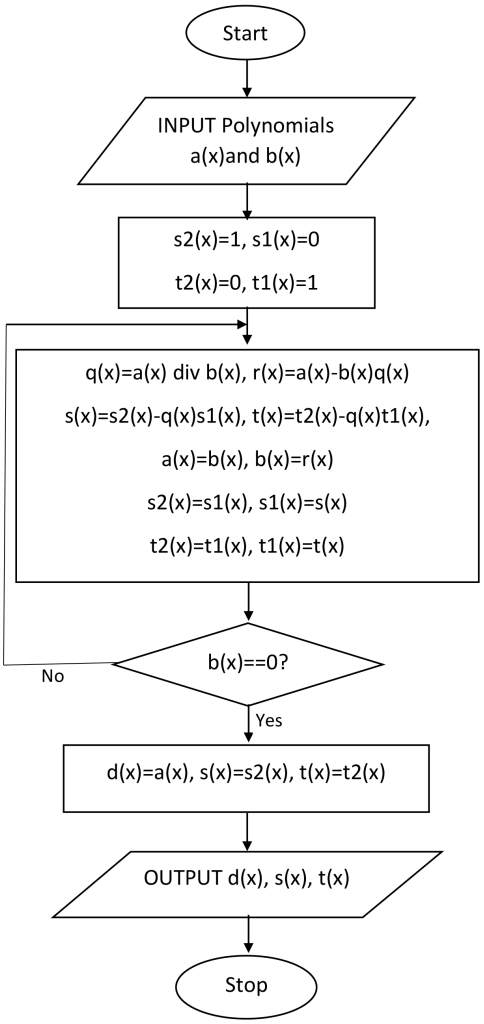

Figure 2. Graphical Representation of Algorithm 2

```
t1x=1;
while(bx!=0)
{
  qx=ax/bx;
  rx=ax-bx*qx;
  sx=s2x-qx*s1x;
  tx=t2x-qx*t1x;
  ax=bx;
  bx=rx;
  s2x=s1x;
  s1x=sx;
  t2x=t1x;
  t1x=tx;
}
dx=ax;
sx=s2x;
tx=t2x;
printf("\nOutput of algorithm 2:
  \nd(x)=gcd(a(x),b(x)) and polynomials
  s(x),t(x) which satisfy s(x)a(x)+t(x)b(x)
  =d(x)\n");
printf("%d=gcd(%d,%d) and polynomials
  %d,%d which satisfy
(%d)*(%d)+(%d)*(%d)=%d\n",dx,a,b,sx,tx,
  sx,a,tx,b,dx);
}
int main()
{
int a[100],b[100];
int i, o_ax, o_bx,ra1;
float x,x_bx, ax,bx;
printf("Enter the value of x for a(x)
  & b(x)\n");
scanf("%f", &x);
printf("Enter the order of the
  polynomial  a(x)\n");
scanf("%d", &o_ax);
printf("Enter %d coefficients \n",
  o_ax + 1);
  for (i = 0; i<= o_ax; i++)
  {
    scanf("%d", &a[i]);
  }
printf("Enter the order of the
```

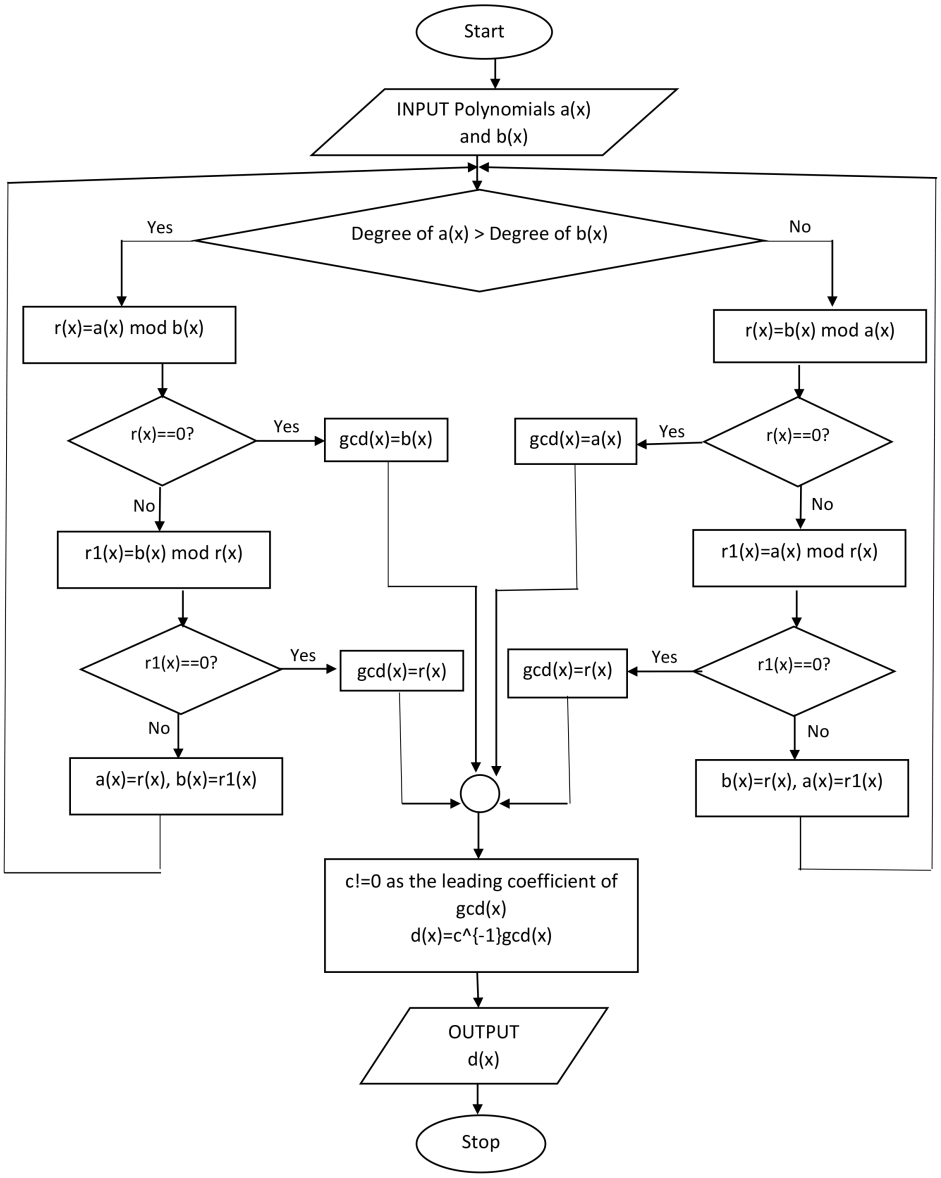

Figure 3. Graphical Representation of Algorithm 3

```
  polynomial b(x)\n");
scanf("%d", &o_bx);
printf("Enter %d coefficients \n",
  o_bx + 1);
for (i = 0; i<= o_bx; i++)
{
  scanf("%d", &b[i]);
}
ax = a[0];
for (i = 1; i<= o_ax; i++)
{
  ax = ax * x+ a[i];
}
printf("The polynomial a(x) = %.0f
 \n", ax);
bx = b[0];
for (i = 1; i<= o_bx; i++)
{
    bx = bx * x + b[i];
  }
  printf("The polynomial b(x) = %.0f
   \n", bx);
    algorithm2(ax,bx);
    return 0;
  }
Output: Figure 6
```

Code for Algorithm 3:

```
\#include<bits/stdc++.h>

using namespace std;

void algorithm3(int ax,intbx,intoax,intobx)
{
   int rx,gcdx,r1x;
   while(oax>obx)
   {
```

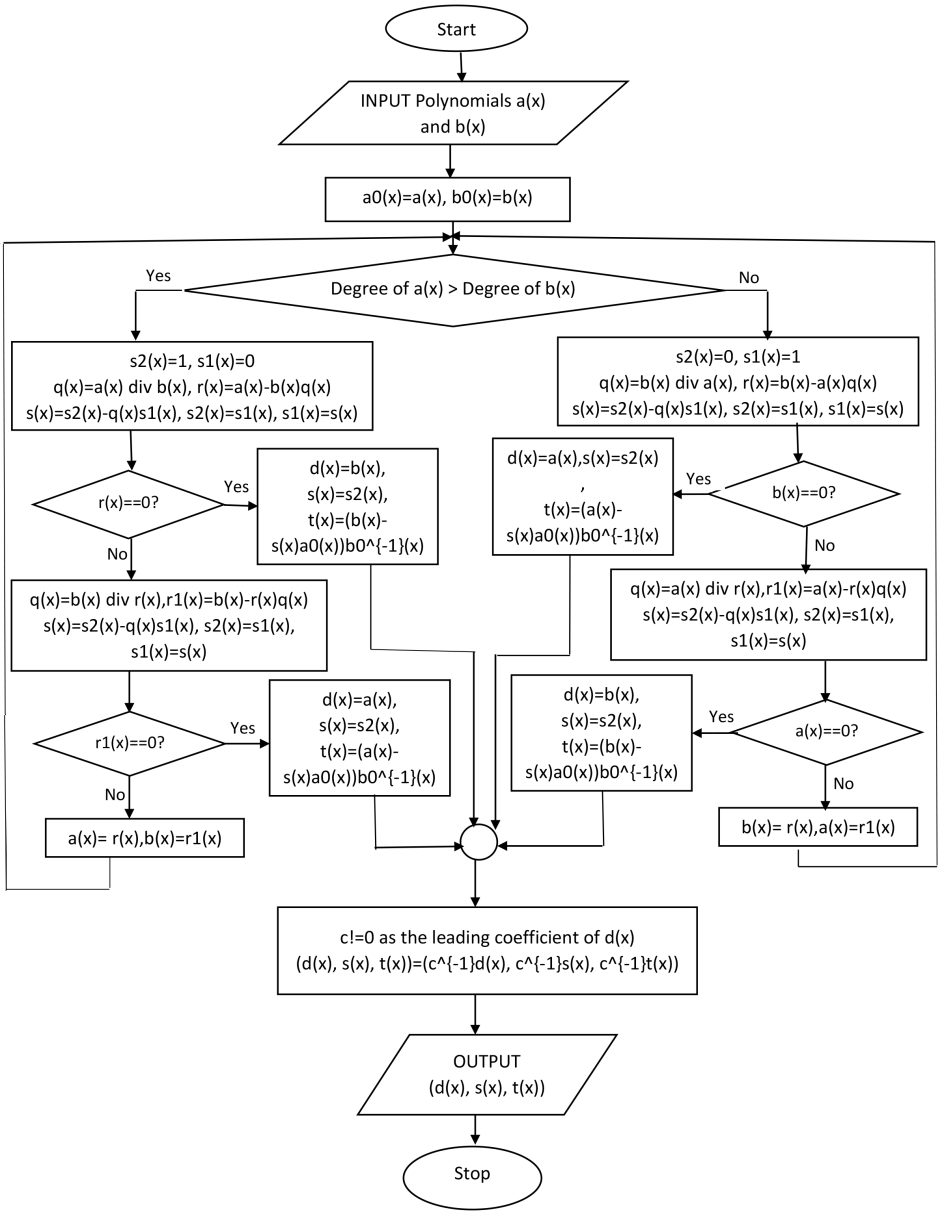

Figure 4. Graphical Representation of Algorithm 4

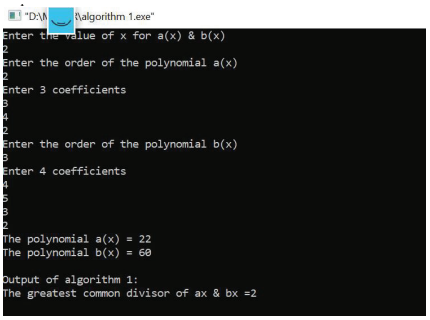

Figure 5. Output of Algorithm 1

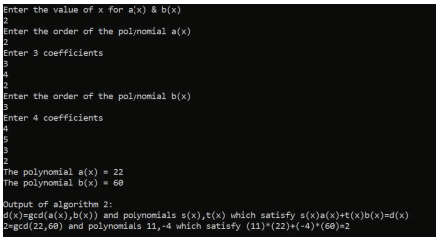

Figure 6. Output of Algorithm 2

```
   rx=ax%bx;
     if(rx==0)
     {
       gcdx=bx;
       break;
     }
   r1x=bx%rx;
     if(r1x==0)
     {
       gcdx=rx;
       break;
     }
   ax=rx;
   bx=r1x;
   }
   while(oax<=obx){
     rx=bx\%ax;
       if(rx==0)
       {
         gcdx=ax;
         break;
       }
     r1x=ax\%rx;
       if(r1x==0)
       {
         gcdx=rx;
         break;
       }
     bx=rx;
     ax=r1x;
     }
   printf("Output of algorithm 3:\n
     %d\n",gcdx);
}
int main()
{
int a[100],b[100];
int i, o_ax, o_bx,ra1;
float x,x_bx, ax,bx;
printf("Enter the value of x for a(x)
 & b(x)\n");
scanf("%f", &x);
printf("Enter the order of the polynomial
 a(x)\n");
scanf("%d", &o_ax);
printf("Enter %d coefficients \n",
 o_ax + 1);
   for (i = 0; i<= o_ax; i++)
   {
     scanf("\%d", &a[i]);
   }
printf("Enter the order of the
   polynomial b(x)\n");
scanf("%d", &o_bx);
printf("Enter %d coefficients \n",
 o_bx + 1);
   for (i = 0; i<= o_bx; i++)
   {
     scanf("%d", &b[i]);
   }
ax = a[0];
   for (i = 1; i<= o_ax; i++)
   {
    ax = ax * x+ a[i];
   }
printf("The polynomial a(x) = %.0f
   \n", ax);
   bx = b[0];
   for (i = 1; i<= o_bx; i++)
   {
     bx = bx * x + b[i];
   }
printf("The polynomial b(x) = %.0f
   \n", bx);
algorithm3(ax,bx,o_ax,o_bx);
return 0;
}
Output: Figure 7
```

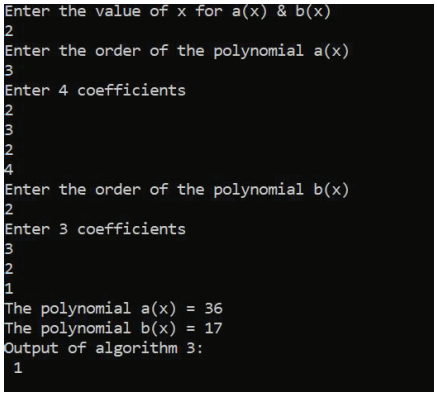

Figure 7. Output of Algorithm 3

Code for Algorithm 4:

```
\#include<bits/stdc++.h>
```

using namespace std;

```
void algorithm4(int ax,intbx,intoax,intobx)
  {
     int aox,box,s2x,s1x,rx,qx,sx;
     aox=ax;
     box=bx;
     if(oax>obx)
     {
       s2x=1;
       s1x=0;
       qx=ax/bx;
       rx=ax-bx*qx;
       sx=s2x-qx*s1x;
       s2x=s1x;
       s1x=sx;
       if(rx==0)
       {
```

```
      dx=bx;
      sx=s2x;
      tx=bx
    }
  }
}
int main()
{
int a[100],b[100];
int i, o_ax, o_bx,ra1;
float x,x_bx, ax,bx;
printf("Enter the value of x for a(x)
  & b(x)\n");
scanf("%f", &x);
printf("Enter the order of the
  polynomial a(x)\n");
scanf("%d", &o_ax);
printf("Enter %d coefficients \n",
  o_ax + 1);
for (i = 0; i<= o_ax; i++)
{
  scanf("%d", &a[i]);
}
printf("Enter the order of the
 polynomial b(x)\n");
scanf("%d", &o_bx);
printf("Enter %d coefficients \n",
 o_bx + 1);
for (i = 0; i<= o_bx; i++)
{
  scanf("%d", &b[i]);
}
ax = a[0];
for (i = 1; i<= o_ax; i++)
{
  ax = ax * x+ a[i];
}
printf("The polynomial a(x) = %.0f
  \n", ax);
bx = b[0];
for (i = 1; i<= o_bx; i++)
{
  bx = bx * x + b[i];
}
printf("The polynomial b(x) = %.0f
  \n", bx);
// algorithm4(ax,bx,o\_ax,o\_bx);
```

```
Enter the value of x for a(x) & b(x)
2
Enter the order of the polynomial a(x)
2
Enter 3 coefficients
3
4
2
Enter the order of the polynomial b(x)
3
Enter 4 coefficients
4
5
3
2
The polynomial a(x) = 22
The polynomial b(x) = 60
```

Figure 8. Output of Algorithm 4

```
   return 0;
}
Output: Figure 8
```

4. Conclusion

This paper shows how easy and efficient we can calculate the gcd of $a(x)$ and $b(x)$ using algorithm 3 and algorithm 4.

References

[1] E. Kaltofen and H. Rolletschek, "Computing greatest common divisors and factorizations in quadratic number fields," *Mathematics of computation*, vol. 53, no. 188, pp. 697–720, 1989.

[2] G. H. Norton, "On the asymptotic analysis of the Euclidean algorithm," *Journal of Symbolic Computation*, vol. 10, no. 1, pp. 53–58, 1990.

[3] A. G. Akritas, "A new method for computing polynomial greates common divisors and polynomial remainder sequences," *Numerische Mathematik*, vol. 52, no. 2, pp. 119–127, 1988.

[4] J. D. Dixon, "The number of steps in the Euclidean algorithm," *Journal of number theory*, vol. 2, no. 4, pp. 414–422, 1970.

[5] H. Rolletschek, "On the number of divisions of the Euclidean algorithm applied to Gaussian integers," *Journal of Symbolic Computation*, vol. 2, no. 3, pp. 261–291, 1986.

[6] A. N. M. R. Karim, F. S. Rafi, M. N. Uddin, M. I. Mahmud, M. U. Khandaker, R. Mahmud, and M. Faruque, "Development of a new algorithm to address the transportation issue," *Civil Engineering and Architecture*, vol. 9, no. 6, pp. 2105–2116, 2021.

[7] A. G. Akritas, G. I. Malaschonok, and P. S. Vigklas, "On the remainders obtained in finding the greatest common divisor of two polynomials," *Serdica Journal of Computing*, vol. 9, no. 2, pp. 123–138, 2015.