Mutant Zhuang-Zi Algorithm

Jintai Ding[1,3,*] and Dieter S. Schmidt[2]

[1] Department of Mathematical Sciences
[2] Department of Computer Science
University of Cincinnati
Cincinnati, OH 45220, USA
[3] Department of Mathematics
Southern Chinese University of Technology
ding@math.uc.edu, dieter.schmidt@uc.edu

Abstract. In this paper we present a new variant of the Zhuang-Zi algorithm, which solves multivariate polynomial equations over a finite field by converting it into a single variable problem over a large extension field. The improvement is based on the newly developed concept of mutant in solving multivariate equations.

Keywords: multivariate polynomials, Hidden Field Equation, polynomial roots, mutant.

1 Introduction

Solving polynomial equations of single or multiple variables has always been a central problem in mathematics. This comes from our desire to understand what is going on with those equations, but more fundamentally it comes from the ubiquitous roles these simple but fundamental problems play in all branches of science and engineering. Though, Babylonians found the first algebraic solution to a single variable quadratic equation [1] more than 3500 years ago, the progress in this area has been very slow. The next successes came 3000 years later with Ferro solving the single variable cubic equation and Ferrari solving the single variable quartic in the 16th century. Galois' theory put an end to finding algebraic formulas for higher order single variable equations.

The situation is much harder in the multivariate case. The real great success came with the Gröbner basis method [2], inspired by ideas of modern algebraic geometry. Recently, a new area of solving multivariate equations over a finite field has attracted a lot of attention, which is inspired by the appearance of multivariate public key cryptography [3]. Here the public key is a set of quadratic polynomials, and in the so called algebraic cryptanalysis one tries to break this cryptosystem by solving a set of multivariate polynomial equations.

Solving a generic set of nonlinear equations over a finite field is not an easy problem, since we know that solving a set of multivariate polynomial equations

[*] The first author was supported by grants from NSF, NSF China and the Taft Foundation of the University of Cincinnati.

over a finite field is, in general, an NP-complete problem [4]. However, much effort has been devoted to search for new methods for solving multivariate polynomial equations over a finite field, along the line of Gröbner bases. Examples include XL [5], the enhanced Gröbner bases methods F_4 and F_5 of Faugère [6,7], the new mutant XL algorithms [8,9,10,11] and the Zhuang-Zi (ZZ) algorithm [12].

Among these algorithms the ZZ algorithm is totally different, since it converts the problem of solving a set of multivariate polynomial equations into solving a polynomial of a single variable over a large extension field. This can be done since any finite n–dimensional vector space over a finite field can be identified as a large finite field of degree n extension over the original finite field. The ZZ algorithm uses the same method as the XL algorithm, except that we try to produce low degree single variable polynomials. Then we solve the low degree polynomial by the very efficient Berlekamp algorithm.

In order to describe the degeneration of multivariate polynomial systems while they are being solved the concept of mutant was introduced recently. It was very successfully applied to improve the XL algorithms [8,9,10,11]. In this paper we will apply the same idea to the ZZ algorithm to produce a new more efficient version of it, which we call the mutant ZZ algorithm.

2 Background

Let k be a finite field with q elements and suppose we have m polynomials $f_0, f_1, \ldots, f_{m-1} \in k[x_0, x_1, \ldots, x_{n-1}]$. We wish to find all $(a_0, a_1, \ldots, a_{n-1}) \in k^n$, such that

$$\begin{aligned} f_0(a_0, a_1, \ldots, a_{n-1}) &= 0 \\ f_1(a_0, a_1, \ldots, a_{n-1}) &= 0 \\ &\vdots \\ f_{m-1}(a_0, a_1, \ldots, a_{n-1}) &= 0 \end{aligned} \tag{1}$$

We may as well work in the ring

$$k[x_0, x_1, \ldots, x_{n-1}]/(x_0^q - x_0, x_1^q - x_1, \ldots, x_{n-1}^q - x_{n-1}),$$

though for convenience we will abuse notation and write $k[x_0, x_1, \ldots, x_{n-1}]$. The key idea of our new algorithm is to shift perspectives from the space of polynomials $k[x_0, x_1, \ldots, x_{n-1}]$ with coefficients in the small field k, to a space of polynomials $K[X]$ with coefficients in some suitably chosen extension field K.

To simplify matters, let us assume that $m = n$. Choose any irreducible polynomial $g(y) \in k[y]$ of degree n. Then $K = k[y]/(g(y))$ is a degree n field extension of k. Let ϕ be the standard k-linear map that identifies K with the n-dimensional vector space k^n, i.e., $\phi : k^n \longrightarrow K$, defined by

$$\phi(a_0, a_1, \ldots, a_{n-1}) = a_0 + a_1 y + \cdots + a_{n-1} y^{n-1} \tag{2}$$

Let $f : k^n \longrightarrow k^n$ be the polynomial map defined by $f = (f_0, f_1, \ldots, f_{n-1})$. We can lift f up to the extension field K using ϕ to create a map $F : K \longrightarrow K$ defined by

$$F = \phi \circ f \circ \phi^{-1}.$$

Using the Lagrangian interpolation formula, we can think of F as a polynomial in $K[X]$, where X is an intermediate. In fact, F has a unique representation in the quotient space $K[X]/(X^{q^n} - X)$. For any given f, the corresponding F can be calculated by solving a set of linear equations. The following theorem tells us the exact form of this representation.

Theorem 1. *Using the notation as defined above, for a linear polynomial map* $f = (f_0, f_1, \dots, f_{n-1})$ *we have*

$$F(X) = \sum_{i=0}^{n-1} \beta_i X^{q^i} + \alpha \mod (X^{q^n} - X),$$

for some $\beta_i, \alpha \in K$. *If* f *is a quadratic polynomial map, then*

$$F(X) = \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} \gamma_{ij} X^{q^i + q^j} + \sum_{i=0}^{n-1} \beta_i X^{q^i} + \alpha \mod (X^{q^n} - X),$$

for some $\gamma_{ij}, \beta_i, \alpha \in K$. *Representations for higher order polynomial maps are similarly described. In the case of* $q = 2$, *the formulas are slightly different.*

In this paper, we will identify the map F with its corresponding representation given in Theorem 1.

It is now clear that we can move freely between multivariate functions and single variable functions, and we will do so in order to solve the original system of equations. This is the basic idea of Matsumoto-Imai, Patarin, Kipnis and Shamir [13,14,15], and is also the basis of our algorithm. Given a system of equations such as (1), the basic strategy will be to lift the associated polynomial map f to the map F in the extension field K. The roots of the representation of F given in Theorem 1 correspond exactly with the solutions to the original system of equations defined over k. Once we have the roots in K, we can descend down to k^n with ϕ^{-1}. It remains to develop techniques for reducing the degree of F, which, if successful, will allow us to use efficient algorithms for solving single variable polynomial equations.

We note a fundamental difference between the ZZ algorithm and others is that the ZZ algorithm can be used only with finite fields and cannot be used with fields of characteristic zero, since the lifting from the multivariate system to a single variable equation works only for a finite field. However, in the case of finite fields, the ZZ algorithm unifies the two problems of solving single variable and multivariate polynomial equations into a single problem. This algorithm is named after Zhuang-Zi, an ancient Chinese philosopher who we believe was one of the first to propose the idea of shifting from a local view of problems to a global view in terms of our mathematical interpretation.

The remainder of this paper is organized as follows. We will explain the basic ZZ algorithm and then our mutant ZZ algorithm. Again, we will present a toy example in order to show how the algorithm works. We then present more meaningful examples and conclude with a discussion of future work.

3 The Zhuang-Zi Algorithm

We will start with the standard case of $m = n$, where we have the same number of variables and equations. The Zhuang-Zi algorithm takes the polynomials $f_0, f_1, \ldots, f_{n-1} \in k[x_0, x_1, \ldots, x_{n-1}]$ and a positive integer D as its input, where D is the upper bound on the degree of a polynomial equation which can be solved efficiently. When successful the algorithm returns all n-tuples $(a_0, a_1, \ldots, a_{n-1}) \in k^n$ such that $f_i(a_0, a_1, \ldots, a_{n-1}) = 0$, for $i = 0, 1, \ldots, n-1$.

– **Step 1:** Choose any degree n irreducible polynomial $g(y) \in k[y]$ and define $K = k[y]/(g(y))$. Let $\phi : k^n \longrightarrow K$ be as defined in (2). Lift the given $f = (f_0, f_1, \ldots, f_{n-1})$ to K by $F = \phi \circ f \circ \phi^{-1}$, and compute the polynomial representation of $F(X)$ modulo $X^{q^n} - X$. If $\deg(F(X)) \leq D$, then go to the last step; otherwise continue to the next step.

– **Step 2:** Let $G = \mathrm{Gal}(K/k)$ be the Galois group of K over k consisting of the Frobenius maps $G_i(X) = X^{q^i}$, for $i = 0, 1, \ldots, n-1$. Calculate

$$F_i(X) = G_i \circ F(X) = F(X)^{q^i} \mod (X^{q^n} - X),$$

for $i = 0, 1, \ldots, n-1$. Note that $F_0(X) = F(X)$.

– **Step 3:** Let N be the total number of monomials that appear in any $F_i(X)$. For each $F_i(X)$ create a row vector in K^N, where the entries are the coefficients of $F_i(X)$ listed in decreasing order, and construct an $n \times N$ matrix using these row vectors. Then use Gaussian elimination to produce a new set of t basis polynomials $S = \{S_0(X), S_1(X), \ldots, S_{t-1}(X)\}$. In other words eliminate the monomials in the order of the highest degree first. Label the elements of S so that $S_{t-1}(X)$ is the element of lowest degree. If $\deg(S_{t-1}(X)) \leq D$, then go to the last step; otherwise continue to the next step.

– **Step 4:** For each $i = 0, 1, \ldots, t-1$ and $j = 0, 1, \ldots, n-1$ compute

$$X^{q^j} S_i(X) \mod (X^{q^n} - X).$$

Amend these polynomials to S. As before, apply Gaussian elimination to the matrix associated with this set of polynomials to produce a set S' of new basis polynomials. Let $S'_{t'-1}(X)$ be the polynomial in S' of minimal degree. If $\deg(S'_{t'-1}(X)) \leq D$, then go to the last step; otherwise replace S with S' and repeat this step.

– **Step 5:** At this point we have a polynomial $\Gamma(x)$ with $\deg(\Gamma(x)) \leq D$. Find the roots of $\Gamma(x) = 0$ with a suitable method to obtain a set $V = \{\alpha \in K \mid \Gamma(\alpha) = 0\}$. The solutions of $F(X) = 0$ will be the subset $\{\alpha \in V \mid F(\alpha) = 0\}$.

Since the complexity of any polynomial root finding method depends on the degree of the given polynomial and the size of the field, so too does the complexity of the Zhuang-Zi algorithm. Improvements in the area of polynomial root finding methods will translate directly into an improvement for the Zhuang-Zi algorithm. Efficient methods for finding the roots of a polynomial in a finite field exist and they are described for example in [16,17,18].

Remark 1. The Zhuang-Zi algorithm works also when $m \neq n$. In this case one has to use the maximum of m and n. When $m < n$, there are fewer equations than variables and one simply introduces $n - m$ polynomials identical to 0. If there are more equations than variables ($m > n$) then one can simply introduce $m - n$ fictitious variables $x_n, \dots, x_{m-1}$.

4 The Mutant Zhuang-Zi Algorithm

Definition 1. *Let X^d with $0 \leq d < q^n$ be the standard basis for the function ring $K[X] \mod (X^{q^n} - X)$. For each monomial X^d define a q-weight as the sum of the coefficients in the q-expansion of the integer d.*

The weight of any polynomial in the ring $K[X] \mod (X^{q^n} - X)$ is the maximal weight of its monomials.

For example, if $q = 3$, the 3-weight of X^{16} is $1 + 2 + 1 = 4$, since

$$16 = 3^2 + 2 \times 3^1 + 1,$$

and the 3-weight of the polynomial $X^{27} + X^{18} + X^{16} + X^{15} + X + 1$ is also 4, since the 3-weight of X^{16} is larger then those of the others. When viewed together with the Frobenius map the weight represents the degree of the original polynomials.

Two procedures for reducing a set of polynomials in X is used several times in the algorithm so that they are defined here in advance:

Definition 2 (Reduce-by-degree(S)). *Let $S=\{S_0(X), S_1(X), \dots, S_{n-1}(X)\}$ be a set of polynomials in X. Let N be the total number of monomials that appear in any element in S. For each element in S, create a row vector in K^N, where the entries are the coefficients of each element listed in decreasing order, and construct an $n \times N$ matrix using these row vectors. Then use Gaussian elimination to produce a new set of t basis polynomials to replace S: $S = \{S_0(X), S_1(X), \dots, S_{t-1}(X)\}$. In other words eliminate the monomials in the order of the highest degree first. Label the elements of S so that $S_{t-1}(X)$ is the element of lowest degree on return from the procedure.*

Definition 3 (Reduce-by-weight(S)). *Let $S'=\{S'_0(X), S'_1(X), \dots, S'_{n-1}(X)\}$ be a set of polynomials in X. Again, let N be the total number of monomials that appear in S'. For each element in S', create a row vector in K^N, where the entries are the coefficients of the elements listed in decreasing order according to the weight (if the same weight, then by the degree), and construct an $n \times N$ matrix using these row vectors. Then use Gaussian elimination to produce and return a new set of t basis polynomials $S' = \{S'_0(X), S'_1(X), \dots, S'_{t-1}(X)\}$. In other words eliminate the monomials in the order of the weight of the degree first, then the degree.*

We now present the mutant ZZ algorithm. Here we will assume $m = n$ but the polynomials in the set $f = (f_0, f_1, \dots, f_{n-1})$ can have different degrees. We first select a degree D, such that we can solve a degree D polynomial over a finite field of size q^n efficiently.

- **Step 1:** Choose any degree n irreducible polynomial $g(y) \in k[y]$ and define $K = k[y]/(g(y))$. Let $\phi : k^n \longrightarrow K$ be as defined in (2). Define $f = (f_0, f_1, \dots, f_{n-1})$, lift this to K by $F = \phi \circ f \circ \phi^{-1}$, and compute the polynomial representation of $F(X)$ modulo $X^{q^n} - X$. If $\deg(F(X)) \leq D$, then go to the last step; otherwise continue to the next step.
- **Step 2:** Let $G = \mathrm{Gal}(K/k)$ be the Galois group of K over k consisting of the Frobenius maps $G_i(X) = X^{q^i}$, for $i = 0, 1, \dots, n-1$. Calculate
$$F_i(X) = G_i \circ F(X) = F(X)^{q^i} \mod (X^{q^n} - X),$$
for $i = 0, 1, \dots, n-1$. Note that $F_0(X) = F(X)$.
Create two sets of polynomials S and S', and both of them consist of $\{F_0, F_1, \dots, F_{n-1}\}$ at the moment.
- **Step 3:** Reduce-by-degree(S). If $\deg(S_{t-1}(X)) \leq D$ then go to the last step.
- **Step 4:** Reduce-by-weight(S'). Create a new set R by copying the polynomials in S'. To each polynomial in R assign the index pair $(w_i, 0)$, where w_i is the weight of the polynomial, and the second value will be referred as the multiplication index, denoted by n_i. The set R will be called the root polynomials.
Create a new set of monomials, which consists of the leading terms of S', and call this set LT. Let W be the weight of the polynomial in R with lowest weight plus 1.
- **Step 5:** Multiply the polynomials R_i of lowest weight by X^{q^j} and amend these polynomials to S' and to S. Change the index of these polynomials in R to (w_i, 1).
- **Step 6:** Reduce-by-degree(S). If $\deg(S_{t-1}(X)) \leq D$ then go to the last step.
- **Step 7:** Reduce-by-weight(S'). Create a new set of monomials, which consists of all leading terms of S', and call this set $\bar{LT}$.
If the number of monomial whose weight is lower than W are the same in both LT and $\bar{LT}$, replace LT by $\bar{LT}$ and go to **Step 8**.
If the number of monomials whose weight is lower than W is different in LT and in $\bar{LT}$, pick the polynomials in S' whose leading terms are the ones with weight lower than W and whose leading terms do not belong to LT. Amend these polynomials to the set R with index (weight of this polynomial, 0).

Then replace LT by $\bar{LT}$, and set W to be the lowest weight of all mutants. The newly amended polynomials in R are the mutants.

- **Step 8:** Set $W = W + 1$. For each root polynomial R_i in R if $w_i + n_i < W$, compute
$$X^d R_i(X) \mod (X^{q^n} - X),$$
where the weight of X^d + w_i=W and amend this polynomial to S and S'. Also increase the multiplication index n_i belonging to R_i by 1. Then go to **Step 6**.
- **Step 9:** At this point there exists a polynomial $\Gamma(x)$ with $\deg(\Gamma(x)) \leq D$. Find the roots of $\Gamma(x) = 0$ with a suitable method to obtain a set $V = \{\alpha \in K \mid \Gamma(\alpha) = 0\}$. The solutions of $F(X) = 0$ will be the subset $\{\alpha \in V \mid F(\alpha) = 0\}$.

5 Examples

We present an illustrative and a toy example to see how the mutant Zhuang-Zi algorithm works in practice. We then present two non-trivial examples where Zhuang-Zi succeeds and Gröbner bases fail.

5.1 An Illustrative Example

Let K be the degree 7 extension of $GF(2)$ given by the irreducible polynomial $y^7 + y + 1$. For the final degree require $D = 1$. Use

$$\begin{aligned}
F(X) &= X^5 + 1 = F_0\\
F^2(X) &= X^{10} + 1 = F_1\\
F^4(X) &= X^{20} + 1 = F_2\\
F^8(X) &= X^{40} + 1 = F_3\\
F^{16}(X) &= X^{80} + 1 = F_4\\
F^{32}(X) &= X^{33} + 1 = F_5\\
F^{64}(X) &= X^{66} + 1 = F_6.
\end{aligned}$$

For this case, the Gaussian elimination is done already and $W = 2$. This set also becomes the root polynomials, each with index pair $(2, 0)$. In Step 5 we add the following new polynomials of weight 3.

$$\begin{aligned}
(X^5 + 1)X &= X^6 + X;\\
(X^5 + 1)X^2 &= X^7 + X^2;\\
(X^5 + 1)X^4 &= X^9 + X^4;\\
(X^5 + 1)X^8 &= X^{13} + X^8;\\
(X^5 + 1)X^{16} &= X^{21} + X^{16};\\
(X^5 + 1)X^{32} &= X^{37} + X^{32};\\
(X^5 + 1)X^{64} &= X^{69} + X^{64}.
\end{aligned}$$

$$\begin{aligned}
(X^{10}+1)X &= X^{11}+X;\\
(X^{10}+1)X^{2} &= X^{12}+X^{2};\\
(X^{10}+1)X^{4} &= X^{14}+X^{4};\\
(X^{10}+1)X^{8} &= X^{18}+X^{8};\\
(X^{10}+1)X^{16} &= X^{26}+X^{16};\\
(X^{10}+1)X^{32} &= X^{42}+X^{32};\\
(X^{10}+1)X^{64} &= X^{74}+X^{64}.
\end{aligned}$$

$$\begin{aligned}
(X^{20}+1)X &= X^{21}+X;\\
(X^{20}+1)X^{2} &= X^{22}+X^{2};\\
(X^{20}+1)X^{4} &= X^{24}+X^{4};\\
(X^{20}+1)X^{8} &= X^{28}+X^{8};\\
(X^{20}+1)X^{16} &= X^{36}+X^{16};\\
(X^{20}+1)X^{32} &= X^{52}+X^{32};\\
(X^{20}+1)X^{64} &= X^{84}+X^{64}.
\end{aligned}$$

$$\begin{aligned}
(X^{40}+1)X &= X^{41}+X;\\
(X^{40}+1)X^{2} &= X^{42}+X^{2};\\
(X^{40}+1)X^{4} &= X^{44}+X^{4};\\
(X^{40}+1)X^{8} &= X^{48}+X^{8};\\
(X^{40}+1)X^{16} &= X^{56}+X^{16};\\
(X^{40}+1)X^{32} &= X^{72}+X^{32};\\
(X^{40}+1)X^{64} &= X^{104}+X^{64}.
\end{aligned}$$

$$\begin{aligned}
(X^{80}+1)X &= X^{81}+X;\\
(X^{80}+1)X^{2} &= X^{82}+X^{2};\\
(X^{80}+1)X^{4} &= X^{84}+X^{4};\\
(X^{80}+1)X^{8} &= X^{88}+X^{8};\\
(X^{80}+1)X^{16} &= X^{96}+X^{16};\\
(X^{80}+1)X^{32} &= X^{112}+X^{32};\\
(X^{80}+1)X^{64} &= X^{17}+X^{64}.
\end{aligned}$$

$$\begin{aligned}
(X^{33}+1)X &= X^{34}+X;\\
(X^{33}+1)X^{2} &= X^{35}+X^{2};\\
(X^{33}+1)X^{4} &= X^{37}+X^{4};\\
(X^{33}+1)X^{8} &= X^{41}+X^{8};\\
(X^{33}+1)X^{16} &= X^{49}+X^{16};\\
(X^{33}+1)X^{32} &= X^{65}+X^{32};\\
(X^{33}+1)X^{64} &= X^{97}+X^{64}.
\end{aligned}$$

$$\begin{aligned}
(X^{66}+1)X &= X^{67}+X;\\
(X^{66}+1)X^{2} &= X^{68}+X^{2};\\
(X^{66}+1)X^{4} &= X^{70}+X^{4};\\
(X^{66}+1)X^{8} &= X^{74}+X^{8};\\
(X^{66}+1)X^{16} &= X^{82}+X^{16};\\
(X^{66}+1)X^{32} &= X^{98}+X^{32};\\
(X^{66}+1)X^{64} &= X^{3}+X^{64}.
\end{aligned}$$

At the beginning of step 7 the root polynomials R with their indices are

$$\begin{aligned} R = \{ & X^{80}+1,\ (2,1);\quad X^{66}+1,\ (2,1);\quad X^{40}+1,\ (2,1);\quad X^{20}+1,\ (2,1); \\ & X^{33}+1,\ (2,1);\quad X^{10}+1,\ (2,1);\quad X^{5}+1,\ (2,1)\ \}. \end{aligned}$$

After Reduce-by-weight(S') the following polynomials with their indices have to be added to R

$$\begin{array}{llllllll} \{X^{96}+X, & (2,0); & X^{72}+X, & (2,0); & X^{68}+X, & (2,0); & X^{48}+X, & (2,0); \\ X^{36}+X, & (2,0); & X^{34}+X, & (2,0); & X^{24}+X, & (2,0); & X^{18}+X, & (2,0); \\ X^{17}+X, & (2,0); & X^{12}+X, & (2,0); & X^{9}+X, & (2,0); & X^{6}+X, & (2,0); \\ X^{3}+X, & (2,0); & X^{64}+X, & (1,0); & X^{32}+X, & (1,0); & X^{16}+X, & (1,0); \\ X^{8}+X, & (1,0); & X^{4}+X, & (1,0); & X^{2}+X, & (1,0)\} & & \end{array}$$

There are six mutants of weight 1. In step 8 we have $W = 2$ and these six mutants are added to S and S' after multiplying each with X, X^2, X^4, X^8, X^{16}, X^{32}, and X^{64}. When we reduce the new system S' we find $X + 1$ and with it the solution to the system.

In the original ZZ, we would have expanded to weight 4 polynomials and solved the system. But here we went only to weight 3 due to the mutant of weight 1. The example as given could have been solved more easily by other methods, but the details were given in order to illustrate how the mutant ZZ algorithm works.

5.2 A Toy Example

In order to show how the algorithm works in a bigger example we use three quadratic equations in three variables with coefficients in the field $k = GF(3)$. We define the polynomial map $f : k^3 \longrightarrow k^3$ by its components

$$\begin{aligned} f_1(x_1,x_2,x_3) &= 2x_1x_2 + x_1 + x_2^2 + 2x_2 + 2x_3 + 1 \\ f_2(x_1,x_2,x_3) &= x_1^2 + x_1x_2 + 2x_2^2 + x_3 \\ f_3(x_1,x_2,x_3) &= x_1^2 + x_1x_2 + 2x_1 + 2x_2^2 + x_3 + 1 \end{aligned}$$

in $k[x_1, x_2, x_3]$.
One irreducible polynomial of degree two with coefficients in k is

$$g(y) = y^3 + 2y + 1.$$

The mapping $\phi : k^3 \longrightarrow K$ is defined by

$$\phi(x_1, x_2, x_3) = x_1 + x_2y + x_3y^2 = X,$$

while $\phi^{-1} : K \longrightarrow k^3$ is defined by (using matrix notation)

$$\phi^{-1}(X) : \quad \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2y^2+1 & 2y^2+2y & 2y^2+y \\ 2y & 2y+1 & 2y+2 \\ 2 & 2 & 2 \end{pmatrix} \begin{pmatrix} X \\ X^3 \\ X^9 \end{pmatrix}$$

With this notation, the polynomial map $F = \phi \circ f \circ \phi^{-1}$ is given by

$$\begin{aligned} F(X) = (2y^2 + y)X^{12} &+ (2y^2 + 2)X^{10} + (2y^2 + 2y)X^9 \\ &+(y^2 + 1)X^6 + (y^2 + y + 2)X^4 + (2y^2 + 2y + 2)X^3 \\ &+(y^2 + 2y + 1)X^2 + (y^2 + 2y + 2)X + y^2 + 1 \end{aligned}$$

Since this is a trivial example, we could factor $F(X)$ directly and obtain

$$\begin{aligned} F(X) = (X &+ y^2 + 2) \\ &(X^{11} + (2y^2 + 1)X^{10} + (y^2 + 1)X^9 + (2y + 2)X^8 + (y^2 + 1)X^7 \\ &+(2y^2 + y + 1)X^6 + (y + 2)X^5 + y^2X^4 + (2y^2 + 2y + 1)X^3 \\ &+2yX^2 + (y + 2)X + y^2 + 2y + 1) \end{aligned}$$

from which we can see that $X = 2y^2 + 1$ is the only solution of the equation $F(X) = 0$ in K and with it $x_1 = 1$, $x_2 = 0$ and $x_3 = 2$.

If we set $D = 1$ we want to find the solution directly in the form of a linear function in X. In the original ZZ algorithm, we need to run the algorithm to the extent such that it will generate a space of full span, namely a space of dimension 26 after Gaussian elimination.

If we run the new mutant in ZZ algorithm, we can solve the equation with a space of dimension only 8. The reason for this is that we do linear combinations of the three quadratic equations, and we actually can find a linear equation inside. This means that the set of equations F_0, F_1 and F_2 will produce an equation of the form

$$X^9 + (y^2 + y + 1)X^3 + (2y^2 + 2)X + 2y + 2 = 0,$$

which can be viewed as a mutant. With the help of this mutant, we can solve the set of equations using only 8 equations such that each equation has only 9 monomial at most. This is great improvement in efficiency compared to the original ZZ algorithm.

Someone may say that we are cheating here since we implicitly have a linear equation inside the set of equations. This is true to certain extent, but the point of this example is to illustrate the advantage of the new mutant ZZ algorithm compared to the original ZZ. In larger examples it is more difficult to demonstrate the concept of a mutant when looking at a larger set of equations.

5.3 Non-trivial Examples

The main application of the ZZ algorithms is to the nonlinear multivariate problem where Gröbner bases methods do not succeed. The Zhuang-Zi algorithm requires that we work in a finite field, whereas Göbner bases do not.

When a Gröbner basis is computed in a finite field, it is accomplished usually by augmenting the original set of equations with those defining the finite field. Examples were constructed in [12], where a set of equations can be solved easily by the Zhuang-Zi algorithm, but only with great difficulties via Gröbner bases.

The basic idea of the construction is to select a function $F(X) : K \longrightarrow K$ of low enough degree, so that it can be factored easily, while the corresponding mapping $f : k^n \longrightarrow k^n$ must be complicated, and therefore difficult to solve.

One such that example is the following case: Let $k = GF(2^3)$ and let $K = k[y]/(g(y))$ be a degree n extension of k, for some irreducible $g(y) \in k[y]$. We use a polynomial of low degree in $K[X]$:

$$F(X) = X^{72} + a_1X^{65} + a_2X^{64} + a_3X^{16} + a_4X^9 + a_5X^8 + a_6X^2 + a_7X + a_8, \quad (3)$$

where the coefficients a_j, for $j = 1, \dots, 8$, are chosen at random from k, treated as a subfield of K via the standard embedding. With $q = 8$, all powers of X in (3) can be written in the form $X^{8^i+8^j}$ or X^{8^i}, and so it is clear that (3) $f = \phi^{-1} \circ F \circ \phi$ is a quadratic polynomial map from k^n to k^n. As in the previous example, it is helpful to write $\phi^{-1} : K \longrightarrow k^n$ using matrix notation

$$\mathsf{A}\,\mathsf{X} = \mathsf{x},$$

where $\mathsf{X} = (X^{8^0}, X^{8^1}, \dots, X^{8^{n-1}})^T$, $\mathsf{x} = (x_0, x_1, \dots, x_{n-1})^T$, and A is an $n \times n$ matrix with entries from K that can easily be found by writing each X^{8^i} as a polynomial in y with coefficients in $k[x_0, x_1, \dots, x_{n-1}]$.

The polynomial (3) can be factored easily by a computer algebra system like Magma [19], depending on the coefficients $a_1, \dots, a_8$ of F, and on the value of n. Finding the corresponding solutions directly with the help of a good Gröbner bases program such as Faugère's F_4 version in Magma [6] requires exponential time with increasing n. However this example does not at all demonstrate the advantage of the new improvement since no computation in adding new polynomials is needed.

In [12] another non-trivial example is presented, where $F(X)$ is of a very high degree and therefore cannot be solved with the simplest form of the Zhuang-Zi algorithm. The example is as follows:

Let $q = 4$, $k = GF(q)$. The multiplicative group for the nonzero elements of this field can be generated by the field element a which satisfies $a^2 + a + 1 = 0$. Take $g(y) \in k[y]$ to be the irreducible polynomial

$$g(y) = y^{12} + y^{11} + ay^{10} + ay^9 + y^8 + y^7 + y^5 + a^2y^4 + ay^3 + a^2y^2 + ay + a,$$

and define $K = k[y]/(g(y))$, a degree $n = 12$ extension of k.

Let $F(X) \in K[X]$ be the polynomial

$$\begin{aligned} F(X) = & a^2X^{17664} + X^{5440} + aX^{5376} + X^{4416} + aX^{4096} + aX^{1360} \\ & + X^{1344} + X^{1280} + a^2X^{1024} + a^2X^{336} + aX^{320} + a^2X^{276} \\ & + X^{85} + aX^{84} + aX^{64} + aX^{21} + X^{20} + a \end{aligned}$$

Here each exponent of X in $F(X)$ is a sum of powers of four. The exponent with the most powers of four is $5440 = 4^3 + 4^4 + 4^5 + 4^6$. Therefore, the components of $f = \phi^{-1} \circ F \circ \phi$ will be of degree four.

The degree of F prevents us from finding the roots of $F(X) = 0$ directly. Also, the F_4 implementation in Magma failed to find a Gröbner basis for $f_0, f_1, \ldots, f_{n-1}$ due to the fact that memory requirements exceeded the available resources on our PC. The Zhuang-Zi algorithm found the polynomial

$$\Gamma(X) = X^{276} + aX^{85} + a^2X^{84} + a^2X^{64} + a^2X^{21} + aX^{20} + a$$

and with it the solutions $\{1, a\}$ of $F(X) = 0$.

6 Discussion and Conclusion

The ZZ algorithm is not just a new algorithm, but a new way to look at the problem of solving a set of multivariate polynomial equations over a finite field. The ZZ algorithm intends to unify the cases of solving multivariate equations with the cases of solving single variable equations.

In this paper we presented an improvement to the original ZZ algorithm inspired by the idea of mutant, which was discovered recently. Our experiments show that there are cases where the mutant Zhuang-Zi algorithm will work much more efficiently than the old one and in a some cases it can beat other algorithms including the Gröbner bases algorithms like F_4.

Currently there are many directions for further research. One particular interesting direction is to understand when the ZZ algorithm has an advantage over other methods. We believe that the ZZ algorithm is well suited for an attack on MPKCs and it should shed new lights on understanding the security of systems like HFE. We intend to explore this in a subsequent paper, where we can further demonstrate the advantage of the new mutant ZZ algorithm over the ZZ algorithm. We have some small scale examples to show this point, but there is still considerable room to improve and optimize the implementation of the algorithm.

References

1. Smith, D.E.: History of Mathematics, vol. 1, 2. Dover, New York (1951-1952)
2. Buchberger, B.: Ein Algorithmus zum Auffinden der Basiselemente des Restklassenrings nach einem nulldimensionalen Polynomideal. Universität Innsbruck (1965)
3. Ding, J., Gower, J., Schmidt, D.: Multivariate Public Key Cryptography. In: Advances in Information Security. Springer, Heidelberg (2006)
4. Garey, M.R., Johnson, D.S.: Computers and intractability. In: A Guide to the theory of NP-completeness. W.H. Freeman, New York (1979)
5. Courtois, N., Klimov, A., Patarin, J., Shamir, A.: Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 392–407. Springer, Heidelberg (2000)
6. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases (F_4). Journal of Pure and Applied Algebra 139, 61–88 (1999)

7. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5). In: International Symposium on Symbolic and Algebraic Computation — ISSAC 2002, July 2002, pp. 75–83. ACM Press, New York (2002)
8. Ding, J.: Mutants and its impact on polynomial solving strategies and algorithms. In: Privately distributed research note, University of Cincinnati and Technical University of Darmstadt, 2006 (2006)
9. Ding, J., Carbarcas, D., Schmidt, D., Buchmann, J., Mohamed, M.S.E., Mohamed, W.S.A.E., Tohaneanu, S., Weinmann, R.P.: Mutant XL. In: SCC 2008 (2008)
10. Mohamed, M.S.E., Mohamed, W.S.A.E., Ding, J., Buchmann, J.: MXL2: Solving Polynomial Equations over GF(2) Using an Improved Mutant Strategy. In: Buchmann, J., Ding, J. (eds.) PQCrypto 2008. LNCS, vol. 5299, pp. 203–215. Springer, Heidelberg (2008)
11. Mohamed, M.S.E., Cabarcas, D., Ding, J., Buchmann, J., Bulygin, S.: MXL_3: An efficient algorithm for computing Gröbner bases of zero-dimensional ideals. In: The 12th International Conference on Information Security and Cryptology (ICISC 2009), Seoul, Korea, December 2009. LNCS, Springer, Heidelberg (2009)
12. Ding, J., Gower, J.E., Schmidt, D.: Zhuang-Zi: A new algorithm for solving multivariate polynomial equations over a finite field. In: PQCrypto 2006: International Workshop on Post-Quantum Cryptography, May 23-26. Katholieke Universiteit Leuven, Belgium (2006)
13. Matsumoto, T., Imai, H.: Public quadratic polynomial-tuples for efficient signature verification and message encryption. In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330, pp. 419–453. Springer, Heidelberg (1988)
14. Patarin, J.: Cryptanalysis of the Matsumoto and Imai public key scheme of Eurocrypt'88. Designs, Codes and Cryptography 20, 175–209 (2000)
15. Kipnis, A., Shamir, A.: Cryptanalysis of the HFE public key cryptosystem by relinearization. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 19–30. Springer, Heidelberg (1999)
16. Geddes, K.O., Czapor, S.R., Labahn, G.: Algorithms for Computer Algebra. Kluwer, Amsterdam (1992)
17. Lidl, R., Niederreiter, H.: Finite Fields, 2nd edn. Encyclopedia of Mathematics and its Application, vol. 20. Cambridge University Press, Cambridge (2003)
18. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra, 2nd edn. Cambridge University Press, Cambridge (2003)
19. Computational Algebra Group, University of Sydney: The MAGMA computational algebra system for algebra, number theory and geometry (2005), `http://magma.maths.usyd.edu.au/magma/`