Practical complexities of probabilistic algorithms for solving Boolean polynomial systems

Stefano Barbero[‡] Emanuele Bellini[†] Carlo Sanna[‡] Javier Verbel[†]

[†]Cryptography Research Centre
Technology Innovation Institute
Abu Dhabi, UAE
emanuele.bellini@tii.ae
javier.verbel@tii.ae

[‡]Department of Mathematical Sciences
Politecnico di Torino
Torino, IT
stefano.barbero@polito.it
carlo.sanna.dev@gmail.com

Abstract

Solving a polynomial system over a finite field is an NP-complete problem of fundamental importance in both pure and applied mathematics. In particular, the security of the so-called multivariate public-key cryptosystems, such as HFE of Patarin and UOV of Kipnis et al., is based on the postulated hardness of solving quadratic polynomial systems over a finite field. Lokshtanov et al. (2017) were the first to introduce a probabilistic algorithm that, in the worst-case, solves a Boolean polynomial system in time $O^*(2^{\delta n})$, for some $\delta \in (0,1)$ depending only on the degree of the system, thus beating the brute-force complexity $O^*(2^n)$. Later, Bjorklund et al. (2019) and then Dinur (2021) improved this method and devised probabilistic algorithms with a smaller exponent coefficient δ.

We survey the theory behind these probabilistic algorithms, and we illustrate the results that we obtained by implementing them in C. In particular, for random quadratic Boolean systems, we estimate the practical complexities of the algorithms and their probabilities of success as their parameters change.

Contents

1 Introduction

Solving a polynomial system

$$p_i(x_1, \ldots, x_n) = 0, \quad i = 1, \ldots, m, \tag{1}$$

of m equations in n unknowns $x_1, \ldots, x_n$ is a fundamental problem in both pure and applied mathematics [11, 32]. Its difficulty depends on the domain of the variables and polynomial coefficients. Over the integers, the problem is undecidable, as a consequence of Matiyasevich's solution of Hilbert's tenth problem [23]. Over an algebraic closed field, determining if the system has a solution is equivalent, by the weak Nullstellensatz, to determining if 1 does not belong to the ideal generated by the polynomials, and for polynomials with rational coefficients establishing ideal membership is known to be exponential space complete [24]. Over a finite field, the case of greatest interest for computer science, the problem is NP-complete already when the polynomials are quadratic [18]. (Note that if all the polynomials are linear then the system can be solved in polynomial time by Gaussian elimination.) Moreover, assuming the exponential time hypothesis [19], there exists no subexponential time (worst-case) algorithm for this problem. This makes it particularly relevant for cryptographic applications. Indeed, the security of the so-called multivariate cryptosystems, such as HFE of Patarin [27] and UOV of Kipnis et al. [21], is based on the postulated hardness of solving quadratic polynomial systems over a finite field. Hereafter, we will focus on the case of the Boolean field $\mathbb{F}_2$, although many results generalize obviously to arbitrary finite fields.

Extremely underdetermined ($m = O(\sqrt{n})$) quadratic polynomial systems can be solved in polynomial time [25], and extremely overdetermined ($n = O(\sqrt{m})$) quadratic polynomial systems are also expected to be solvable in polynomial time [10]. For quadratic polynomial systems having $m = n$ and satisfying certain algebraic assumptions, which have been experimentally verified to hold for most of random systems, Bardet et al. [2] proposed a $O^*(2^{0.792n})$ expected-time algorithm. Furthermore, the Yang–Chen variant [34] of the XL algorithm of Courtois et al. [10] solves quadratic polynomial systems with $m = n$ in time $O^*(2^{0.875n})$, but again requires specific algebraic assumptions. These methods inspired the "crossbreed" algorithm of Joux and Vitse [20], which has a fast implementation on GPUs [26] (for a study of the complexity of the crossbreed algorithm, see [14]).

Solving a polynomial system can also be done by computing a Gröbner basis of the ideal generated by the equations, and efficient algorithms for computing Gröbner bases include Faugère's F4 [15] and F5 [16], and their several variants (as Hybrid-F5 [5]). However, the asymptotic complexities of these algorithms are not well-understood. In general, the complexity of the Gröbner basis computation can be estimated by $O\big(\mathrm{Mon}(d_{\mathrm{reg}})^\omega\big)$, where d_{reg} is the so-called *degree of regularity*, $\mathrm{Mon}(d_{\mathrm{reg}})$ is the number of monomials of degree not exceeding d_{reg}, and ω is the linear algebra constant [1, 17] (see also [3]).

Lokshtanov et al. [22] were the first to introduce a probabilistic algorithm that solves a square ($m = n$) polynomial system in time $O^*(2^{\delta n})$, for some $\delta < 1$ depending only on the degree of the system, without relying on any unproved assumption. In particular, for quadratic systems their algorithm has complexity $O^*(2^{0.8765n})$. They used the so-called "polynomial method", which replaces the whole system of equations by a single random polynomial whose truth table can be computed more efficiently than by brute force. These ideas were improved by Björklund et al. [7], who introduced a parity-counting argument that reduced the complexity to $O^*(2^{0.804n})$, and then by Dinur [13], who considered a multiparity-counting argument and reduced further the complexity to $O^*(2^{0.6943n})$. Furthermore, Dinur [12] devised a probabilistic algorithm that, under some reasonable assumptions, solves random quadratic polynomial systems in time $O(n^2 \cdot 2^{0.815n})$. Note that this last complexity is asymptotically worse than the previous algorithm of Dinur (and also Björklund et al.), however it does not hide polynomial factors and the hidden constant is expected to be "small".

The purpose of this paper is twofold:

1. Survey the new probabilistic algorithms for solving Boolean polynomial systems, in order to provide a useful resource for researchers interested in these new methods.

2. Provide experimental data about how actual implementations of these new algorithms perform in practice, including running times and probabilities of success.

Regarding the second point, the last two authors wrote an implementation in C [29] of the probabilistic algorithms, and tested its performance on random Boolean polynomial systems. We did so motivated by the well-known fact that the asymptotic complexity of an algorithm can be misleading when it comes to applications. Indeed, an algorithm with a better asymptotic complexity can be outperformed by an algorithm with a worse asymptotic complexity when they run on real world data, due to large hidden constants in the asymptotic notation and/or very different worse-case and average-case behaviors. A classic example is Coppersmith–Winograd algorithm for matrix multiplication [9] that, despite being asymptotically better than other matrix-multiplication algorithms like Strassen's [31], it is never used in practice because of the huge hidden constant in its asymptotic complexity. Moreover, authors usually introduce a new algorithm by providing a high-level explanation that, while simplifying the exposition, could hide technical difficulties that may arise in producing an actual implementation and that may increase the actual complexity. Therefore, after the theoretical analysis, implementations and experiments are still necessary to understand the real performance of any new algorithm.

The paper is structured as follows: in Section 2 we provide the mathematical preliminaries; in Section 3 we explain the logic of the probabilistic algorithms; in Sections 4 and 5 we illustrate our implementations of the algorithms and the experimental results; and in Section 6 we summarize our general conclusions.

Acknowledgements

S. Barbero and C. Sanna are members of GNSAGA of INdAM and of CrypTO, the group of Cryptography and Number Theory of Politecnico di Torino.

2 Preliminaries

2.1 Notation

We employ Landau's notation $f(n) = O(g(n))$, with its usual meaning that $|f(n)| \leq C|g(n)|$ for some constant $C > 0$. Also, we write $f(n) = O^*(g(n))$ whenever $f(n) = O(n^k g(n))$ for some constant $k \geq 0$. We let log denote the logarithm in base 2, and $|\mathcal{A}|$ denote the cardinality of every set $\mathcal{A}$. We reserve the letters x, y, z, w for formal variables. We indicate with $\mathbb{F}_2$ the field with two elements, and with $\mathbb{F}_2^n$ its n-fold Cartesian product. For every $a, b \in \mathbb{F}_2^n$, we write $a \leq b$ to mean that $a_i \leq b_i$ for $i = 1, \dots, n$, and we let $|a|$ denote the *Hamming weight* of a, that is, the number of $i \in \{1, \dots, n\}$ such that $a_i = 1$. Also, we define $\mathcal{W}_w^n := \{a \in \mathbb{F}_2^n : |a| \leq w\}$ for every $w \leq n$.

2.2 Boolean functions

We recall some basic facts on Boolean functions. A *Boolean function* is a map $f : \mathbb{F}_2^n \to \mathbb{F}_2$, and its *support* is $\operatorname{supp}(f) := f^{-1}(1)$. Once an order on $\mathbb{F}_2^n$ is fixed, we have that f is completely described by its *truth table* $[f(a) : a \in \mathbb{F}_2^n]$. Furthermore, f has a unique representation as a *Boolean polynomial*, that is, an element of the quotient ring

$$\mathbb{F}_2[x_1, \dots, x_n] \,/\, (x_1^2 - x_1, \dots, x_n^2 - x_n).$$

This representation is called the *algebraic normal form (ANF) of f*. Precisely, the ANF of f is

$$f(x) = \sum_{a \in \mathbb{F}_2^n} \zeta[f](a)\, x_1^{a_1} \cdots x_n^{a_n},$$

where $\zeta[f] : \mathbb{F}_2^n \to \mathbb{F}_2$ is the *zeta transform of f*, defined by

$$\zeta[f](a) := \sum_{b \leq a} f(b), \tag{2}$$

for every $a \in \mathbb{F}_2^n$. Given the truth table of f, computing $\zeta[f]$ using (2) requires $O(3^n)$ additions. A more efficient algorithm, usually attributed to Yates [35], provides a way to compute $\zeta[f]$ using only $O(n2^n)$ additions. More generally, if $\mathcal{A} \subseteq \mathbb{F}_2^n$ is a *downward closed set*, that is, if $a \in \mathcal{A}$ implies that $b \in \mathcal{A}$ for every $b \in \mathbb{F}_2^n$ with $b \leq a$, then the values of $\zeta[f]$ over $\mathcal{A}$ are completely determined by the values of f over $\mathcal{A}$, and they can be computed by Yates's algorithm using $O(n|\mathcal{A}|)$ additions, see Algorithm 1.

Algorithm 1: Yates's algorithm for computing the zeta transform.

function ZetaTransform($[f(a) : a \in \mathcal{A}]$)
  input : The partial truth table $[f(a) : a \in \mathcal{A}]$ of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ over a downward closed set $\mathcal{A} \subseteq \mathbb{F}_2^n$.
  output: The zeta transform $\left[\zeta[f](a) : a \in \mathcal{A}\right]$.
  `// This is an in-place algorithm:` $[f(a) : a \in \mathcal{A}]$ `is overwritten.`
  for $i = 1, \ldots, n$ **do**
    for $a \in \mathcal{A}$ **do**
      if $a_i = 1$ **then**
        $f(a) \leftarrow f(a) + f(a_1, \ldots, a_{i-1}, 0, a_{i+1}, \ldots a_n)$
  return $[f(a) : a \in \mathcal{A}]$

Thus the zeta transform converts the truth table of a Boolean function to the ANF. It turns out that the zeta transform is its own inverse, meaning that $\zeta[\zeta[f]] = f$ for every f, and consequently it also provides a way back from the AFN to the truth table. When used in this way, it is also known as the *Möbius transform*. In particular, if f is a Boolean function with $\operatorname{supp}(\zeta[f]) \subseteq \mathcal{A}$, where $\mathcal{A} \subseteq \mathbb{F}_2^n$ is a downward closed set, then the knowledge of the values of f over $\mathcal{A}$ is enough to determine all the values of f, a process known as *interpolation*, see Algorithm 2. An important case is that in which f has ANF of degree d, so that $\operatorname{supp}(\zeta[f]) \subseteq \mathcal{W}_d^n$, and consequently f is completely determined by its values over $\mathcal{W}_d^n$.

Algorithm 2: Interpolation of a Boolean function.

function Interpolation($[f(a) : a \in \mathcal{A}], \mathcal{B}$)
  input : The partial truth table $[f(a) : a \in \mathcal{A}]$ of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ with $\operatorname{supp}(\zeta[f]) \subseteq \mathcal{A}$, and two downward closed sets $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathbb{F}_2^n$.
  output: The partial truth table $[f(a) : a \in \mathcal{B}]$.
  $[g(a) : a \in \mathcal{A}] \leftarrow$ ZetaTransform($[f(a) : a \in \mathcal{A}]$)
  Define $[\tilde{g}(a) : a \in \mathcal{B}]$ by $\tilde{g}(a) := g(a)$ for $a \in \mathcal{A}$, and $\tilde{g}(a) := 0$ for $a \notin \mathcal{A}$.
  return ZetaTransform($[\tilde{g}(a) : a \in \mathcal{A}]$)

2.3 Solving a Boolean polynomial system

Hereafter, let $p_1, \ldots, p_m \in \mathbb{F}_2[x_1, \ldots, x_n]$ be polynomials of degree at most d, given by their ANFs. The problem of solving the polynomial system (1) has three important versions:

- Decisional: the output is **True** if (1) has a solution, and **False** otherwise;

- Search: the output is any (single) solution of (1), if it is solvable, and **Null** otherwise;
- Exhaustive: the output is the whole set of solutions of (1).

The decisional and search versions are strictly related. On the one hand, obviously, solving the search version also solves the decisional. On the other hand, by iteratively testing each variable, one can solve the search version by calling a subroutine for the decisional version at most n times, see Algorithm 3.

Algorithm 3: Search using Decisional.

function Search$(p_1, \ldots, p_m)$
  input : Polynomials $p_1, \ldots, p_m \in \mathbb{F}_2[x_1, \ldots, x_n]$.
  output: A solution of $\{p_i(x) = 0\}_{i=1}^m$ if it exists, and **Null** otherwise.
  $(a_1, \ldots, a_n) \leftarrow (0, \ldots, 0)$
  for $i = 1, \ldots, n$ **do**
    $(q_1, \ldots, q_m) \leftarrow (p_1|_{x_i=0}, \ldots, p_m|_{x_i=0})$
    `//` Decisional$(q_1, \ldots, q_m)$ `returns True if the system` $\{q_i(x) = 0\}_{i=1}^m$ `has a solution, and False otherwise.`
    if Decisional$(q_1, \ldots, q_m)$ **then**
      $a_i \leftarrow 0$
      $(p_1, \ldots, p_m) \leftarrow (q_1, \ldots, q_m)$
    else
      $a_i \leftarrow 1$
      $(p_1, \ldots, p_m) \leftarrow (p_1|_{x_i=1}, \ldots, p_m|_{x_i=1})$
  if $(p_1, \ldots, p_m) = (0, \ldots, 0)$ **then**
    return $(a_1, \ldots, a_n)$
  else
    return Null

2.4 Razborov–Smolensky construction

In all the probabilistic algorithms we will examine, the polynomial

$$F(x) := \prod_{i=1}^{m}(1 + p_i(x)) \tag{3}$$

is considered, in order to set up a decisional version of the problem of solving (1), or in order to reduce it to a parity-counting problem as we will explain next. Indeed, $a \in \mathbb{F}_2^n$ is a solution of (1) if and only if $F(a) = 1$. Since in general $\deg(F) = dm$, the ANF of F may be too large to be manipulated. Thus (3) is approximated by a probabilistic polynomial of smaller degree, using the following construction credited to Razborov [28] and Smolensky [30].

Let $\ell \in \{1, \ldots, m\}$ be a parameter. For $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$ pick $\alpha_{ij} \in \mathbb{F}_2$ uniformly at random, and define the polynomials

$$R_i(x) := \sum_{j=1}^{m} \alpha_{ij} p_j(x), \quad i = 1, \ldots, \ell. \tag{4}$$

Note that, for every $a \in \mathbb{F}_2^n$ and $i \in \{1, \ldots, \ell\}$, if $F(a) = 1$, then $R_i(a) = 0$; whereas if $F(a) = 0$, then there exists j such that $p_j(a) = 1$, and consequently $\Pr[R_i(a) = 0] = \frac{1}{2}$. Therefore, defining

$$\tilde{F}(x) := \prod_{i=1}^{\ell}(1 + R_i(x)), \tag{5}$$

it follows that $\Pr\left[\tilde{F}(a) = F(a)\right] \geq 1 - 2^{-\ell}$ for every $a \in \mathbb{F}_2^n$, that is, $\tilde{F}$ approximates F with high probability (depending on ℓ). Moreover, $\deg(\tilde{F}) \leq d\ell$, which can be much lower than the degree of F.

2.5 Valiant–Vazirani affine hashing

The Valiant–Vazirani affine hashing [33] is a probabilistic method to isolate an unique solution of the system (1). It consists on adding a set of random linear equations to (1) in this way: If we assume that $\mathcal{S} \subseteq \mathbb{F}_2^n$ is the nonempty set of solutions to (1) and $k = 0, 1, 2, \ldots, n$ is the unique integer such that $2^k \leq |\mathcal{S}| < 2^{k+1}$, we draw independent and uniform random values $\alpha_{i,j}, \beta_i \in \mathbb{F}_2$ for $i = 1, 2, \ldots, k+2$, $j = 1, 2, \ldots, n$ and add the linear equations

$$\sum_{j=1}^{n} \alpha_{i,j} x_j = \beta_j, \quad i = 1, 2, \ldots k+2 \tag{6}$$

to (1). For the probability $\Pr[U_x]$ that $x \in \mathcal{S}$ is the only solution to (1) which also satisfies (6) the following inequality holds (see [7, Section 2.5])

$$\Pr[U_x] \geq \frac{1}{2^{k+3}}.$$

Therefore

$$\Pr[\cup_{x \in \mathcal{S}} U_x] = \sum_{x \in \mathcal{S}} \Pr[U_x] \geq \frac{1}{8}.$$

From this fact and thanks to the inequalities $\left(1 - \frac{1}{8}\right)^r \leq e^{-\frac{r}{8}} \leq \varepsilon$, with $r = \lceil 8 \ln \varepsilon^{-1} \rceil$ independent repetitions of this procedure we can isolate a unique solution $x \in \mathcal{S}$ with probability $1 - \varepsilon$. Since k is unknown we can consider $\varepsilon = \frac{1}{n}$ and exhaustively try out all the values $k = 0, 1, \ldots, n$. Thus using this method, if (1) is solvable, a solution can be isolated with high probability with $O(n \log n)$ steps.

3 Probabilistic algorithms

A *probabilistic algorithm* is an algorithm that employs a source of randomness in some of its steps, and which is known to return the correct answer with a probability bounded from below, usually depending on some parameters of the algorithm. Some authors use the term *randomized algorithm* (of the Monte Carlo type), but we avoided it because it seem to suggest that a deterministic algorithm was somehow "randomized", which is not the case of the probabilistic algorithms we are considering.

3.1 Lokshtanov et al.'s algorithm

The main result of Lokshtanov et al. [22] for polynomial systems over $\mathbb{F}_2$ is the following:

Theorem 3.1. *A polynomial system over $\mathbb{F}_2$ of degree d and n unknowns can be solved by a probabilistic algorithm in time $O^*\big(2^{0.8756n}\big)$ for $d = 2$, and $O^*\big(2^{(1-1/(5d))n}\big)$ for $d > 2$.*

Lokshtanov et al.'s probabilistic algorithm solves the decisional version of the problem. Their idea is to put $n_1 := \lfloor \delta n \rfloor$, where $\delta \in (0, 1)$ is a parameter depending only on d, and split the variables $x_1, \dots, x_n$ into $y := x_1, \dots, x_{n-n_1}$ and $z := x_{n-n_1+1}, \dots, x_n$, so that the polynomial (3) can be written as $F(y, z)$. Then, letting

$$V := \bigvee_{a \in \mathbb{F}_2^{n-n_1}} \left(\sum_{b \in \mathbb{F}_2^{n_1}} s_b F(a, b) \right), \tag{7}$$

where $s_b \in \mathbb{F}_2$ are draw randomly with uniform distribution, we have that: If (1) has no solution, then $V = 0$; otherwise, if (1) has solutions, then $V = 1$ with probability at least $1/2$.

Computing V directly from (7) is too inefficient, because of the large degree of $F(y, z)$. Therefore, $F(y, z)$ is replaced by a random polynomial $\tilde{F}(y, z)$ of lower degree using the Razborov–Smolensky construction, as explained in Section 2.4. In this case, it is chosen $\ell := n_1 + 2$. Moreover, the truth table of the polynomial

$$\sum_{b \in \mathbb{F}_2^{n_1}} s_b \tilde{F}(y, b) \in \mathbb{F}_2[y]$$

is computed efficiently using Yates's algorithm for the Möbius transform.

In order to correct the errors introduced by approximating $F(y, z)$ with $\tilde{F}(y, z)$, it can be proved that it is enough to repeat this whole procedure $100n$ times, and then return the result that occurs at least $40n$ times.

Finally, a careful complexity analysis [22, Lemma 3.3] shows that the best choice for the parameter is $\delta = 0.1235$ for $d = 2$, and $\delta = 1/(5d)$ for $d > 2$, which leads to the claimed time complexities. For a pseudocode see Algorithm 4.

3.2 Björklund et al.'s algorithm

The main result of Björklund et al. [7] for polynomial systems over $\mathbb{F}_2$ is the following:

Theorem 3.2. *A polynomial system over $\mathbb{F}_2$ of degree d and n unknowns can be solved by a probabilistic algorithm in time $O^*\big(2^{0.804n}\big)$ for $d = 2$, and $O^*\big(2^{(1-1/(2.7d))n}\big)$ for $d > 2$.*

The key idea of Björklund et al. is to introduce the *parity-counting problem* for the system (1), which is the problem of determining the parity of the number of solutions of (1), that is, 0 if the number of solutions if even, and 1 if the number of solutions is odd. In light of the previous considerations on the polynomial (3), this amount to computing the sum

$$\sum_{c \in \mathbb{F}_2^n} F(c). \tag{8}$$

Algorithm 4: Lokshtanov et al.'s algorithm.

```
function LokAlgo(p_1, ..., p_m)
    input : p_1, ..., p_m ∈ F_2[x_1, ..., x_n] of degree at most d, given by their ANFs.
    output: True if {p_i(x) = 0}_{i=1}^m has a solution, False otherwise.
    n_1 ← ⌊δn⌋ // δ := 0.1235 for d = 2, and δ := 1/(5d) for d > 2.
    ℓ ← n_1 + 2
    s ← 100n
    Initialize array [Score(a) : a ∈ F_2^{n−n_1}] of integers to zeros.
    for t = 1, 2, ..., s do
        // Below y = x_1, ..., x_{n−n_1}
        S(y) ← 0 ∈ F_2[y]
        for b ∈ F_2^{n_1} do
            R_i(y) ← Σ_{j=1}^m α_{i,j} p_j(y, b) for i = 1, ..., ℓ, where α_{i,j} ∈ F_2 are random.
            F̃(y) ← Π_{i=1}^ℓ (1 + R_i(y))
            S(y) ← S(y) + s_b F̃(y) where s_b ∈ F_2 is random.
        T ← ZetaTransform(ANF of S(y), F_2^{n−n_1}) // Truth table of S(y),
            computed by the Möbius transform.
        for a ∈ F_2^{n−n_1} do
            if T(a) = 1 then
                Score(a) ← Score(a) + 1
    for a ∈ F_2^{n−n_1} do
        if Score(a) > 40n then
            return True
    return False
```

Once (at most) one solution of (1) has been isolated using the Valiant–Vazirani affine hashing, solving the parity-counting problem is equivalent to determining if the system has a solution.

The evaluation of (8) is performed using the probabilistic approximation $\tilde{F}$ of F defined in (5), which has a lower degree than F. Similarly to Loskshtanov et al., the variables $x_1, \ldots, x_n$ are split into $y := x_1, \ldots, x_{n-n_1}$ and $z := x_{n-n_1+1}, \ldots, x_n$, where $n_1 := \lfloor \lambda n \rfloor$ and $\lambda \in (0,1)$ is a parameter depending only on the degree d, so that F can be written as $F(y,z)$. Defining

$$G(y) := \sum_{b \in \mathbb{F}_2^{n_1}} \tilde{F}(y,b) = \sum_{b \in \mathbb{F}_2^{n_1}} \prod_{i=1}^{\ell} (1 + R_i(y,b)) \tag{9}$$

we have that

$$\sum_{c \in \mathbb{F}_2^n} \tilde{F}(c) = \sum_{a \in \mathbb{F}_2^{n-n_1}} G(a). \tag{10}$$

In order to evaluate (9), note that $\deg(G) \leq d\ell - n_1$, and consequently G can be interpolated by computing its values on $\mathcal{W}_{d\ell-n_1}^{n-n_1}$. Since for any $a \in \mathcal{W}_{d\ell-n_1}^{n-n_1}$ we have

$$G(a) = \sum_{b \in \mathbb{F}_2^{n_1}} \prod_{i=1}^{\ell} (1 + R_i(a,b)),$$

this can be intended as a parity-counting problem for the polynomial system

$$R_i(a,z) = 0, \quad i = 1, \ldots, \ell,$$

with degree $dn_1 - \ell$. Therefore, interpolating G is equivalent to $|\mathcal{W}_{d\ell-n_1}^{n-n_1}|$ recursive calls for solving (smaller) parity-counting instances.

It is possible to show that for each $a \in \mathbb{F}_2^{n-n_1}$ it holds

$$\Pr\left[G(a) = \sum_{b \in \mathbb{F}_2^{n_1}} F(a,b)\right] \geq 1 - 2^{n_1 - \ell},$$

and choosing $\ell = n_1 + 2$ the computed partial parity is correct with probability at least $3/4$.

Finally, for each $a \in \mathbb{F}_2^{n-n_1}$ the error is reduced by computing s approximations of each partial parity via independent probabilistic polynomials $\{G_k(y)\}_{k=1}^{s}$ and using a scoreboard of votes, selecting as correct parity the one that appears more than $s/2$ times. Taking $s = 48n+1$, this majority vote for each $a \in \mathbb{F}_2^{n-n_1}$ across all s approximations gives the parity with an exponentially small probability of error.

An analysis of the complexity [7, Section 3.7] shows that the best choice for the parameter is $\lambda = 0.1967\ldots$ for $d = 2$, and $\lambda = 1/(2.7d)$ for $d > 2$, which leads to the time complexities of Theorem 3.2. For a pseudocode see Algorithm 5.

3.3 Dinur's algorithm

Dinur [13] improved the results of Björklund et al. with the following theorem:

Algorithm 5: Björklund et al. algorithm.

Function Parity($\{p_i(x)\}_{i=1}^m$)**:**

  input : $p_1, \ldots, p_m \in \mathbb{F}_2[x_1, \ldots, x_n]$ of degree at most d, given by their ANFs.

  output: The parity P of the number of solutions of the system $\{p_i(x) = 0\}_{i=1}^m$.

  $n_1 \leftarrow \lfloor \lambda n \rfloor$ `//` $\lambda := 0.1967$ `for` $d = 2$`, and` $\lambda := 1/(2.7d)$ `for` $d > 2$`.`

  $\ell \leftarrow n_1 + 2$

  $s \leftarrow 48n + 1$

  Initialize array $[Score(a) : a \in \mathbb{F}_2^{n-n_1}]$ of integers to zeros.

  for $k = 1, \ldots, s$ **do**

    `// Below` $y = y_1, \ldots, y_{n-n_1}$ `and` $z = z_1, \ldots, z_{n_1}$`.`

    $R_i(y, z) \leftarrow \sum_{j=1}^m \alpha_{i,j} p_j(y, z)$ for $i = 1, \ldots, \ell$, where $\alpha_{i,j} \in \mathbb{F}_2$ are random.

    Initialize the array $[V(a) : a \in \mathbb{F}_2^{n-n_1}]$ of Booleans to zeros.

    for $a \in \mathcal{W}_{d\ell-n_1}^{n-n_1}$ **do**

      $V(a) \leftarrow V(a) + \mathsf{Parity}(\{R_h^{(k)}(a, z)\}_{h=1}^{\ell})$

    $T \leftarrow \mathsf{Interpolation}\big([V(a) : a \in \mathbb{F}_2^{n-n_1}], \mathbb{F}_2^{n-n_1}\big)$ `// Truth table of` $G^{(k)}(y)$ `over` $\mathbb{F}_2^{n-n_1}$`.`

    $Score \leftarrow Score + T$ `// Update the score with componentwise sum.`

  $P \leftarrow 0$

  for $a \in \mathbb{F}_2^{n-n_1}$ **do**

    if $Score(a) > s/2$ `// Majority vote.`

    then

      $P \leftarrow P + 1$

  return P

Theorem 3.3. *A polynomial system over $\mathbb{F}_2$ of degree d and n unknowns can be solved by a probabilistic algorithm in time $O^*\big(2^{0.6943n}\big)$ for $d = 2$, and $O^*\big(2^{(1-1/(2d))n}\big)$ for $d > 2$. Moreover, there exists a probabilistic algorithm that outputs all the K solutions. For an arbitrarily small $\varepsilon > 0$, the runtime of this algorithm is $O^*\big(\max\big(2^{0.6943n}, 2^{\varepsilon n}K\big)\big)$ for $d = 2$, and $O^*\big(\max\big(2^{(1-1/(2d))n}, 2^{\varepsilon n}K\big)\big)$ for $d > 2$.*

The main idea of Dinur's algorithm relies in the observation that all the smaller parity-counting instances are related as they originate from the same system.

Definition 3.1. Given as input polynomials $p_1, \dots, p_m \in \mathbb{F}_2[x_1, \dots, x_n]$ in their ANFs, and nonnegative integers $n_1 \leq n$ and $w \leq n - n_1$, the *multiple parity-counting problem* asks to determine the array $[V(a) : a \in \mathcal{W}_w^{n-n_1}]$ such that

$$V(a) = \sum_{b \in \mathbb{F}_2^{n_1}} F(a, b), \qquad \text{for } a \in \mathcal{W}_w^{n-n_1}, \tag{11}$$

where $F(y, z)$ is polynomial (3) in which $x = (y_1, \dots, y_{n-n_1}, z_1, \dots, z_{n_1})$ after a partition of the variables.

As in Björklund et al. [7], in order to evaluate (11) the approximation (9) using probabilistic polynomials is considered and its evaluation is performed reducing via recursion an instance of the multiple parity counting problem to only a few instances of the same problem. Dinur's reduction does not fix any variable to a particular value, but rather changes internal parameters of the multiple parity-counting instance which determine the number of inner parity-counting instances and the number of variables over which each parity is computed. The finer control over the parameters of the induced instances allows for a potentially more efficient self-reduction with respect to the one used in [7]. For a pseudocode of Dinur's multiple parity counting algorithm see Algorithm 6.

Another novelty in Dinur's algorithm is the approach used in isolating solutions. The Valiant–Vazirani affine hashing isolates only one solution at a time, and applying it to obtain all solutions will be inefficient unless their number is small. Dinur introduced a method that isolates and outputs many solutions "in parallel". Say that a solution $(\sigma, \tau) \in \mathbb{F}_2^{n-n_1} \times \mathbb{F}_2^{n_1}$ to the system (1) is *isolated* with respect to the partition $(y_1, \dots, y_{n-n_1}, z_1, \dots, z_{n_1})$ if for any $\tau' \neq \tau$, we have that (σ, τ') is not a solution. The solutions are isolated and outputted performing r times a linear change of variables with uniform invertible matrices in $\mathbb{F}_2^{n \times n}$, then finding all the isolated solutions to the equivalent systems with $n_1 + 1$ calls to the multiple parity counting algorithm. Indeed, for a fixed partition of the new variables, assuming that all $\sigma \in \mathbb{F}_2^{n-n_1}$ for which the returned parity is 1 correspond to isolated solutions, the n_1 remaining bits of τ can be recovered one at a time with n_1 additional calls to the multiple parity counting algorithm, where in the ith call z_i is fixed to 0. In order to avoid "false positive" every candidate solution is tested to control if it is indeed a solution to the system. Choosing $r = 2n$ and a suitable value for n_1, this procedure outputs all the isolated solutions with negligible probability of error [13, Section 4.1]. For a pseudocode of this procedure see Algorithm 7.

With the same value of $n_1 < n$ the multiple parity counting algorithm begins in a similar way to the previous related algorithms in [22] and [7], by choosing a parameter ℓ, considering the probabilistic polynomial (5) and defining a first partition of the variables

Algorithm 6: Dinur MultParityCount.

function MultParityCount($\{p_h(y,z)\}_{h=1}^{m}, n_1, w$)

  input : $\{p_h(y,z)\}_{h=1}^{m}$ polynomials of degrees $d \geq 2$ in the variables $y = (y_1, \ldots, y_{n-n_1})$, $z = (z_1, \ldots, z_{n_1})$, represented in ANF, n_1, w.

  output: The solution $[V(a) : a \in \mathcal{W}_w^{n-n_1}]$ of the multiparity-counting problem (see Definition 3.1).

  $n_2 \leftarrow \lfloor n_1 - \lambda n \rfloor$ `//` $\lambda \in (0,1)$ `is a parameter.`

  if $n_2 \leq 0$ **then**

    return BruteForceMultParity($\{p_h(y,z)\}_{h=1}^{m}, n_1, w$)

  $\ell \leftarrow n_2 + 2$

  $s \leftarrow 48n + 1$

  Initialize the array $[Score(c) : c \in \mathcal{W}_w^{n-n_1} \times \mathbb{F}_2^{n_1-n_2}]$ of integers with zeros.

  for $k = 1, \ldots, s$ **do**

    $R_i(y,z) \leftarrow \sum_{j=1}^{m} \alpha_{i,j} p_j(y,z)$ for $i = 1, \ldots, \ell$, where $\alpha_{i,j} \in \mathbb{F}_2$ are random.

    `// Below` $u = z_1, \ldots, z_{n_1-n_2}$ `and` $v = z_{n_1-n_2+1}, \ldots, z_{n_2}$`.`

    $V_1 \leftarrow$ MultParityCount($\{R_i((y,u),v)\}_{i=1}^{\ell}, n_2, d \cdot \ell - n_2$)

    $e \leftarrow$ Interpolation($[V_1(a) : a \in \mathcal{W}_{d\cdot\ell-n_2}^{n-n_2}], \mathcal{W}_w^{n-n_1} \times \mathbb{F}_2^{n_1-n_2}$)

    $Score \leftarrow Score + e$ `// Update the score with componentwise sum.`

  Initialize the array $[V(a) : a \in \mathcal{W}_w^{n-n_1}]$ of Booleans with zeros.

  for $a \in \mathcal{W}_w^{n-n_1}$ **do**

    for $b \in \mathbb{F}_2^{n_1-n_2}$ **do**

      if $Score(a,b) > s/2$ `// Majority vote`

      then

        $V(a) \leftarrow V(a) + 1$

  return V

Algorithm 7: Dinur ExhaustSolutions.

function $\mathsf{ExhaustSolutions}(\{p_h(x)\}_{h=1}^m, n_1)$

  input : $p_1, \ldots, p_m \in \mathbb{F}_2[x_1, \ldots, x_n]$ of degree at most d, given by their ANFs, n_1.

  output: All the solutions to the system $\{p_h(x) = 0\}_{h=1}^m$.

  $r \leftarrow 2n$

  for $k = 1, \ldots, r$ **do**

    Draw random invertible matrix $B \in \mathbb{F}_2^{n \times n}$

    $\{q_h(v)\}_{h=1}^m \leftarrow$ Change of variables $v = B^{-1}x$ in $\{p_h(x)\}_{h=1}^m$.

    Initialize array $[M(i, a) : i \in \{0, \ldots, n_1\}, a \in \mathbb{F}_2^{n-n_1}]$ of Booleans to zeros.

    `// Below` $y = v_1, \ldots, v_{n-n_1}$ `and` $z = v_{n-n_1+1}, \ldots, v_n$`.`

    $M(0, \cdot) \leftarrow \mathsf{MultParityCount}(\{q_h(y, z)\}_{h=1}^m, n_1, n - n_1)$

    for $i = 1, \ldots, n_1$ **do**

      $M(i, \cdot) \leftarrow$
      $\mathsf{MultParityCount}(\{q_h(y, z_1, \ldots, z_{i-1}, 0, z_{i+1}, \ldots, z_{n_1})\}_{h=1}^m, n_1 - 1, n - n_1)$

    for $a \in \mathbb{F}_2^{n_1}$ **do**

      if $M(0, a) = 1$ **then**

        $sol \leftarrow a$

        for $i = 1, \ldots, n_1$ **do**

          if $M(i, a) = 1$ **then**

            $sol \leftarrow sol \parallel 0$

          else

            $sol \leftarrow sol \parallel 1$

      if $\{p_h(B^{(k)} \cdot sol) = 0\}_{h=1}^m$ **then**

        return $B^{(k)} \cdot sol$

$x = (y_1, \ldots, y_{n-n_1}, z_1, \ldots z_{n_1})$. Then an additional partition on z is done with another parameter $n_2 < n_1$ obtaining $z = (u, v) = (u_1, \ldots, u_{n_1-n_2}, v_1, \ldots, v_{n_2})$.

Considering

$$G(y,u) = \sum_{c \in \mathbb{F}_2^{n_2}} \tilde{F}(y,u,c) = \sum_{c \in \mathbb{F}_2^{n_2}} \prod_{i=1}^{\ell} (1 + R_i(y,u,c)),$$

the evaluation of every partial parity

$$G(a,b) = \sum_{c \in \mathbb{F}_2^{n_2}} \tilde{F}(a,b,c)$$

for $a \in \mathbb{F}_2^{n-n_1}$ and $b \in \mathbb{F}_2^{n_1-n_2}$ is equivalent to a parity counting instance of the system

$$R_1(a,b,v) = 0, \ldots R_\ell(a,b,v) = 0,$$

and we have

$$\Pr\left[G(a,b) = \sum_{c \in \mathbb{F}_2^{n_2}} F(a,b,c)\right] \geq 1 - 2^{n_2-\ell},$$

which is at least $\frac{3}{4}$ fixing $\ell = n_2 + 2$. Since $\deg(G) \leq d\ell - n_2$ in order to interpolate G it is sufficient to compute its values in $\mathcal{W}_{d\ell-n_2}^{n-n_2}$. The main difference from the Björklund et al. parity counting algorithm is in the way that the $|\mathcal{W}_{d\ell-n_2}^{n-n_2}|$ valuations of G are computed. In Dinur's algorithm all the $|\mathcal{W}_{d\ell-n_2}^{n-n_2}|$ parity counting instances are performed as a single recursive call of the multiple parity-counting algorithm, where in the nested calls brute force evaluation is used only when the parameter defining the partition becomes less or equal to 0. For a pseudocode of the bruteforce algorithm employed by Dinur see Algorithm 8. The multiple parity-counting algorithm outputs the vector in $\mathcal{W}_{d\ell-n_2}^{n-n_2}$ whose entries for all

Algorithm 8: Dinur BruteforceMultParity.

function $\mathsf{BruteForceMultParity}(\{p_h(y,z)\}_{h=1}^{m}, n_1, w)$
  input : $\{p_h(y,z)\}_{h=1}^{m}$ polynomials of degrees $d \geq 2$ in the variables $y = (y_1, \ldots, y_{n-n_1})$, $z = (z_1, \ldots, z_{n_1})$, represented in ANF, n_1, w.
  output: The solution $[V(a) : a \in \mathcal{W}_w^{n-n_1}]$ of the multiparity-counting problem (see Definition 3.1).
  Initialize the array $[e(c) : c \in \mathcal{W}_w^{n-n_1} \times \mathbb{F}_2^{n_1}]$ of Booleans with all ones. ;
  for $k = 1, \ldots, m$ **do**
    $p^{(k)} \leftarrow \mathsf{ZetaTransform}(\text{ANF of } 1 + p_k(y,z), \mathcal{W}_w^{n-n_1} \times \mathbb{F}_2^{n_1})$ `// ` $p^{(k)}$ `is the truth table of` $1 + p_j(y,z)$ `over` $\mathcal{W}_w^{n-n_1} \times \mathbb{F}_2^{n_1}$`.`
    $e \leftarrow e \wedge p^{(k)}$ `// Bitwise AND the evaluation.`
  return $\left[\sum_{b \in \mathbb{F}_2^{n_1}} F(a,b) : a \in \mathcal{W}_w^{n-n_1}\right]$

$(a,b) \in W_{d\ell-n_2}^{n-n_2} \times \mathbb{F}_2^{n_1-n_2}$ are the parities $G(a,b)$ used to interpolate $G(y,u)$. Since the multiple parity counting algorithm calls itself with parameter $n_2' < n_2$, the number of variables over which the polynomials are defined increases with the recursion depth, but their degree $n_2'(d-1)+2d$ decreases (since $d-1 \geq 1$). Moreover

$$\left|\mathcal{W}^{n-n_2}_{n_2(d-1)+2d}\right| > \left|\mathcal{W}^{n-n_2'}_{n_2'(d-1)+2d}\right|$$

so the new instance is not harder than the original one, ensuring an efficient self–reduction. As stated before, the correct partial parity is obtained with probability at least $\frac{3}{4}$ and the error correction is performed via scoreboarding and majority vote on the $s = 48n+1$ approximations given by the polynomials $\{G_k(y,u)\}_{k=1}^{s}$ for all $(a,b) \in \mathcal{W}^{n-n_1}_{w} \times \mathbb{F}_2^{n_1-n_2}$ in order to obtain the true partial parity and the output vector of partial parities with exponentially small probability of error.

3.4 Dinur's second algorithm

Dinur presented a second algorithm in [12], essentially designed for a cryptographic setting, whose complexity estimate can be resumed in the following statement

Statement 1. *Under some reasonable assumptions, a polynomial system of m degree d equations selected at random in n variables over $\mathbb{F}_2$ can be solved (up to small constants) with a running time of $O\left(n^2 \cdot 2^{0.815n}\right)$ if $d = 2$ and $O\left(n^2 \cdot 2^{(1-1/(2.7d))n}\right)$ if $d > 2$.*

The basic idea of this algorithm relies on the observation that in order to find a solution to the system (1) it is sufficient to consider the probabilistic polynomials (4), enumerate isolated solutions to the system

$$R_i(x_1,\ldots,x_n) = 0, \quad i = 1,\ldots,\ell, \quad \ell < m \tag{12}$$

because the set of solutions to (12) is a superset of the solution set of (1), then test each isolated solution on (1). In Dinur [12] Proposition 3.1 shows that for a variable partition $x = (y,z) = (y_1,\ldots,y_{n-n_1},z_1,\ldots,z_{n_1})$ where $n_1 = \ell - 1$ and assuming that (σ,τ) is an isolated solution to (1) the following inequality holds

$$\Pr\left[(\sigma,\tau)\,\text{is an isolated solution to (12)}\,\right] \geq 1 - 2^{n_1-\ell} = \frac{1}{2}$$

so assuming that (1) has an isolated solution this solution is also isolated for (12) with probability at least 1/2. Another important assumption is needed: the system (1) must have an isolated solution with high probability. However in a cryptographic setting, given a variable partition (y,z) grouping a solution to (1) together with $2^{n_1}-1$ different assignments, it is reasonable that each such assignment satisfies (1) with probability 2^{-m}. Thus a solution to (1) is isolated with probability at least $1-2^{n_1-m}$, which is very closed to 1, since usually $m \gg n/5$ and in this algorithm n_1 is chosen such that $n_1 < n/5$ in order to optimize the complexity. For a pseudocode of Dinur's second algorithm see Algorithm 9 As in previous Dinur's algorithm [13] isolated solutions are recovered bit-by-bit by computing n_1+1 sums, but enumerating isolated solutions of (12) rather than the ones of (1). Exploiting the low degree $d_{\tilde{F}}$ of the polynomial (5) the algorithm interpolates

$$U_0\left(y\right) = \sum_{b\in\mathbb{F}_2^{n_1}} \widetilde{F}\left(y,b\right) \quad \text{and} \quad U_i\left(y\right) = \sum_{b\in\mathbb{F}_2^{n_1-1}} \widetilde{F}_{|b_i=0}\left(y,b\right) \quad \text{for} \quad i = 1,\ldots,n_1,$$

Algorithm 9: Dinur Solve.

function Solve($\{p_h(x)\}_{h=1}^m, N$)

input : $\{p_h(x)\}_{h=1}^m$ polynomials of degrees $d \geq 2$ in the variables $x = (x_1, \ldots, x_n)$, represented in ANF, N maximum number of tries.

output: A solution to the system $\{p_h(x) = 0\}_{h=1}^m$ or Failure if after N tries nothing has been found.

Parameters: n_1, $d_{\tilde{F}}$

Initialization: $\ell \leftarrow n_1 + 1$, $w \leftarrow d_{\tilde{F}}$

$PotSolsList \leftarrow NewList()$ // $PotSolsList$ is an $N \times 2^{n-n_1} \times \ell$ multidimensional array: for $k = 0, \ldots, N-1$ $PotSolsList[k]$ is a $2^{n-n_1} \times \ell$ matrix storing the values of $CurPotSols$ where for $i = 0, \ldots, 2^{n-n_1} - 1$, $CurPotSols[i] = (U_0(a^{(i)}), \ldots, U_{n_1}(a^{(i)}))$ and $a^{(i)}$ is the i-th element of $\mathbb{F}_2^{n-n_1}$ according to a predefinite order

$c \leftarrow 0$, $k \leftarrow 0$

while $c = 0 \wedge k \leq N - 1$ **do**

  Draw a uniformly random matrix $\left[\alpha_{i,j}^{(k)}\right] \in \mathbb{F}_2^{\ell \times m}$ of full rank ℓ and compute $\{R_i^{(k)}(x) = \sum_{j=1}^m \alpha_{i,j}^{(k)} p_j(x)\}_{i=1}^\ell$

  $CurPotSols \leftarrow$ OutputPotSols($\{R_i^{(k)}(x)\}_{i=1}^\ell, n_1, w$)

  $PotSolsList[k] \leftarrow CurPotSols$

  if $k \neq 0$ **then**

    $i \leftarrow 0$

    while $c = 0 \wedge i \leq 2^{n-n_1} - 1$ **do**

      if $CurPotSols[i][0] = 1$ // test if $CurPotSols[i]$ is valid

      then

        // if $CurPotSols[i]$ has been otput before test if $sol = a^{(i)}||CurPotSols[i]$ is a solution to the original system

        for $k_1 = 0, \ldots, k-1$ **do**

          if $CurPotSols[i] = PotSolsList[k_1][i]$ **then**

            $sol \leftarrow a^{(i)}||CurPotSols[i]$

            if $\{p_j(sol) = 0\}_{j=1}^m$ **then**

              $c \leftarrow c + 1$

              return sol

            else

              break // continue with next i

      $i \leftarrow i + 1$

  $k \leftarrow k + 1$

if $c = 0$ **then**

  print Failure

where for an isolated solution (σ, τ)

$$U_0(\sigma) = 1, \quad U_i(\sigma) = \tau_i + 1 \quad \text{for} \quad i = 1, \ldots, n_1,$$

and then evaluates $\{U_i(y)\}_{i=0}^{n_1}$ on all $a \in \mathbb{F}_2^{n-n_1}$ to recover isolated solutions. For a pseudocode of the algorithm that outputs the potential isolated solutions see Algorithm 10. Thanks to Proposition 3.3 Dinur [12] shows that these interpolations can be optimized considering the solutions to (12) in the set $\mathcal{W}_{d_{\tilde{F}}-n_1+1}^{n-n_1} \times \mathbb{F}_2^{n_1}$, using, in order to find these solutions, the exhaustive search algorithm of Bouillaguet et al. [8]. For a pseudocode of this interpolation procedure see Algorithm 11. Therefore the computation of the sums $\sum_{b \in \mathbb{F}_2^{n_1}} \widetilde{F}(a, b)$ is only needed , instead of evaluating the sums $\sum_{b \in \mathbb{F}_2^{n_1}} F(a, b)$ which are too expensive to compute directly due to the high degree of F. In the previous algorithms of Björklund et al. [7] and Dinur [13], such sums are computed by majority voting across $48n + 1$ evaluations of different polynomials, a method which is no more used here, reducing the complexity of the algorithm by a factor of $\Omega(n)$. Once that an isolated solution to (12) is found it has to be tested to control if it is a solution to (1). However, these tests make expensive evaluations of polynomials, which generally require about $|\mathcal{W}_d^n|$ bit operations. This may give rise to a large overhead, in particular for $d > 2$. Thus, to avoid this, the algorithm is repeated a certain small number of times. In each of its iterations different sets of independent probabilistic polynomials (4) are used and only the candidate solutions that are output more than once are tested, under the assumption, based on the randomness assumptions about the input system, that it is unlikely for an incorrect candidate solution to be suggested more than once.

4 Implementation

The last two authors wrote an implementation in C of the probabilistic algorithms [29]. The implementation is self-contained, depending only on the C standard library. Its core modules are:

- `bvar.c`, `bfunc.c`, which implement the basic algorithms for Boolean functions. In particular, Yates's algorithm for the zeta transform;
- `bpoly.c`, which implement symbolic computation with Boolean polynomials (uses in Lokshtanov et al.'s algorithm);
- `qsyst.c`, `qpoly.c`, which implement the data structures and the basic functions for quadratic polynomial systems.

The remaning modules implement the probabilistic algorithms. More details are provided in the comments of the source code.

Although we did not focus very much on optimizations, we tried to take advantage of the binary architecture of the processor. For instance, the elements of $\mathbb{F}_2^n$ are stored as 64-bits machine words (under the assumption $n \leq 64$) and, when possible, operations between them are performed using machine words instructions (bitwise xor, or, and, shift, popcount...). In principle, this reduces the complexity by a factor of n. In particular, we used Gosper's

Algorithm 10: Dinur OutputPotSols.

function OutputPotSols($\{R_i(x)\}_{i=1}^{\ell}, n_1, w$)

  input : $\{R_i(x)\}_{i=1}^{\ell}$ probabilistic polynomials of degrees $d \geq 2$ in the variables $x = (x_1, \ldots, x_n)$, represented in ANF, n_1, w.

  output: A $2^{n-n_1} \times \ell$ matrix Out where $Out[i] = (U_0(a^{(i)}), \ldots, U_{n_1}(a^{(i)}))$ and $a^{(i)}$ is the i–th element of $\mathbb{F}_2^{n-n_1}$ according to a predefinite order.

  Do a partition of the variables: $x = (y, z) = (y_1, \ldots, y_{n-n_1}, z_1, \ldots, z_{n_1})$

  $(V, ZV) \leftarrow$ ComputeUValues($\{R_i(y, z)\}_{i=1}^{\ell}, n_1, w$) `//` $V \in \mathbb{F}_2^{|\mathcal{W}_w^{n-n_1}|}$ `is the truth table of` $U_0(y)$ `over` $\mathcal{W}_w^{n-n_1}$ `and` $ZV \in \mathbb{F}_2^{n_1 \times |\mathcal{W}_{w+1}^{n-n_1}|}$ `is a matrix whose` i`--th row is the truth table of` $U_i(y)$ `over` $\mathcal{W}_{w+1}^{n-n_1}$

  $u^{(0)} \leftarrow$ Interpolation($V, \mathbb{F}_2^{n-n_1}$)

  for $i = 1, \ldots, n_1$ **do**

    $u_i \leftarrow$ Interpolation($ZV[i], \mathbb{F}_2^{n-n_1}$)

  `//` $u^{(i)} \in \mathbb{F}_2^{n-n_1}$ `stores the coefficents of` $U_i(y)$ `in ANF obtained via interpolation`

  $Evals \leftarrow 0$ `//` $Evals$ `is a` $n_1 + 1 \times 2^{n-n_1}$ `matrix initialized to` 0 `such that` $Evals[i][j] = U_i(a^{(j)})$ `where` $a^{(j)}$ `is the` j`--th element of` $\mathbb{F}_2^{n-n_1}$ `according to a predefinite order`

  for $i = 0, \ldots, n_1$ **do**

    $Evals[i] \leftarrow$ ZetaTransform($u^{(i)}$)`// evaluate the truth table of` $U_i(y)$ `over` $\mathbb{F}_2^{n-n_1}$

  $Out \leftarrow 0$ `// initialize the output matrix` $Out \in \mathbb{F}_2^{2^{n-n_1} \times n_1+1}$ `to` 0

  for $j = 0, \ldots 2^{n-n_1} - 1$ **do**

    if $Evals[0][j] = 1$ `// check if` $U_0(a^{(j)}) = 1$`, i. e., if` $a^{(j)}$ `could be part of a potential solution`

    then

      $Out[j][0] \leftarrow 1$

      for $i = 1, \ldots, n_1$ **do**

        $Out[j][i] \leftarrow Evals[i][j] + 1$ `// copy potential solution by flipping evaluation bit, since for a potential solution` $(a^{(j)}, b)$ `we have` $Evals[i][j] = U_i(a^{(j)}) = b_i + 1$

  return Out

Algorithm 11: DinurComputeUValues

function $\mathsf{ComputeUValues}(\{R_i(y,z)\}_{i=1}^{\ell}, n_1, w)$

  input : $\{R_i(y,z)\}_{i=1}^{\ell}$ probabilistic polynomials of degrees $d \geq 2$ in the variables $(y,z) = (y_1, \ldots, y_{n-n_1}, z_1, \ldots, z_{n_1})$, represented in ANF, n_1, w.

  output: $V \in \mathbb{F}_2^{|\mathcal{W}_w^{n-n_1}|}$ the truth table of $U_0(y)$ over $\mathcal{W}_w^{n-n_1}$ and $ZV \in \mathbb{F}_2^{n_1 \times |\mathcal{W}_{w+1}^{n-n_1}|}$ a matrix whose i–th row is the truth table of $U_i(y)$ over $\mathcal{W}_{w+1}^{n-n_1}$.

  $Sols \leftarrow \mathsf{BruteForceSystem}(\{R_i(y,z)\}_{i=1}^{\ell}, n-n_1, w+1)$ `//` `Sols is a` $L \times n$ `matrix such that the` i`-th row` $Sol[i]$ `stores the` i`-th solution to the system` $\{R_i(y,z)=0\}_{i=1}^{\ell}$ `found with bruteforce on` $\mathcal{W}_{w+1}^{n-n_1} \times \mathbb{F}_2^{n_1}$`, supposing that this system has` L `solutions`

  $V \leftarrow 0$, $ZV \leftarrow 0$ `// initializaton of the truth tables for all the` $U_i(y)$, $i = 0, \ldots, n_1$

  for $i = 1, \ldots, L$ **do**

    $a \leftarrow Sols[i][1, \ldots, n-n_1]$

    $b \leftarrow Sols[i][n-n_1+1, \ldots, n]$ `//` $(y,z) = (a,b)$ `is the` i`--th solution`

    if $|a| \leq w$ `// values of the Hamming weight` $|a|$ `exceeding` w `do not contribute to` $U_0(y)$

    then

      $j \leftarrow \mathsf{IndexOf}(a, n-n_1, w)$ `// get the index of` a `in` $\mathcal{W}_w^{n-n_1}$ `according to a predefinite ordering`

      $V[j] \leftarrow V[j] + 1 \bmod 2$

    for $k = 1, \ldots, n_1$ **do**

      if $b_k = 0$ **then**

        $j \leftarrow \mathsf{IndexOf}(a, n-n_1, w+1)$

        $ZV[k][j] \leftarrow ZV[k][j] + 1 \bmod 2$

  return (V, ZV)

algorithm [4, Item 175] to loop efficiently through the elements of $\mathcal{W}_w^n$ by using only few instructions.

As source of randomness, which is essential to all the probabilistic algorithms, we found that using the pseudorandom generator of the C standard library (linear feedback shift register) is enough.

We remark that implementing these probabilistic algorithms presents a difficulty due to their probabilistic nature. Precisely, while, for example, in an implementation of an algorithm for Gröbner basis computation, one can run the implementation and check that at every step the computation is consistent; For these probabilistic algorithms one cannot do the same, since each step has a certain amount of randomness, and only at the end of the computation the theory ensures that the probability of error is sufficiently small.

5 Experimental results

In this section, we illustrate our experimental results about the practical complexities of the probabilistic algorithms. We decided to measure the practical complexity in terms of the number of clock cycles taken by the algorithms. Of course, other choices are possible. For example, measuring the time of execution. Nevertheless, different choices give results that are essentially proportional to each other and do not change our general conclusions, since we are mostly interested in the growth rate of the practical complexity.

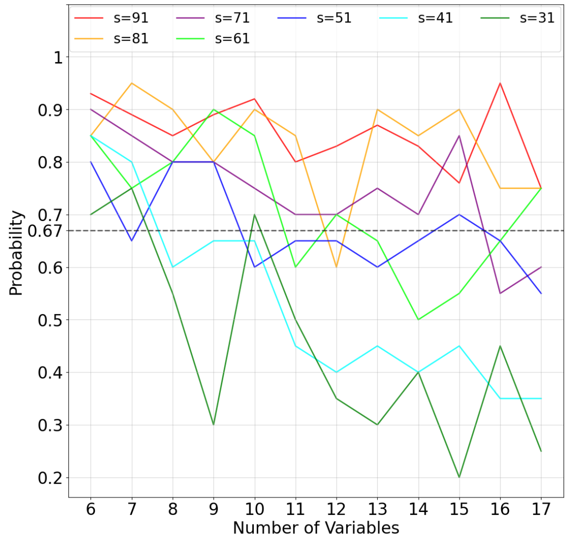
In all our experiments, we randomly generated square ($m = n$, that is, same number of equations and variables) quadratic systems that have a unique solution. We did so because if P is a system having at most one solution, then determining the consistency of P is equivalent to determining the parity of the number of solutions of P. So that, in this particular case, the subroutine Decisional used in the algorithm Search (3) can be replaced by one of the three algorithms LPTWY (4), BKW (5), and MultParityCount (6)[1]. We call the resulting algorithms Search-LPTWY, Search-BKW, and Search-Dinur1.

All these algorithms use an internal loop of size s, which is chosen high enough to guarantee an overwhelming probability of success in n. For LPTWY it is set $s = 100n$, while for BKW and MultParityCount the choice is $s = 48n + 1$. In practice, we can significantly reduce this value and keep a reasonably high probability of success in the algorithms Search-LPTWY, Search-BKW, and Search-Dinur1. For a given n and several values of s, we estimated the probability of success of the algorithms Search-LPTWY, Search-BKW, and Search-Dinur1 on solving a square system of size n with s internal repetitions. The results of such estimations are shown in Figures 1(a), 1(b), and 1(c), respectively. In each case, we ran 100 samples to estimate such a probability[2]. Based on these estimations, we approximate the minimum value s_{min} of s such that Search effectively finds a solution with probability greater than 2/3, see Table 1. Overall, for Search-BKW and Search-Dinur1 we have $s_{min} \approx 3n$, while for Search-LPTWY this number grows roughly as $6n$. In practice, the choice of only s_{min} internal repetitions represents a speed up by a factor of 15 for each algorithm, with at least a probability of 2/3 of finding a solution.

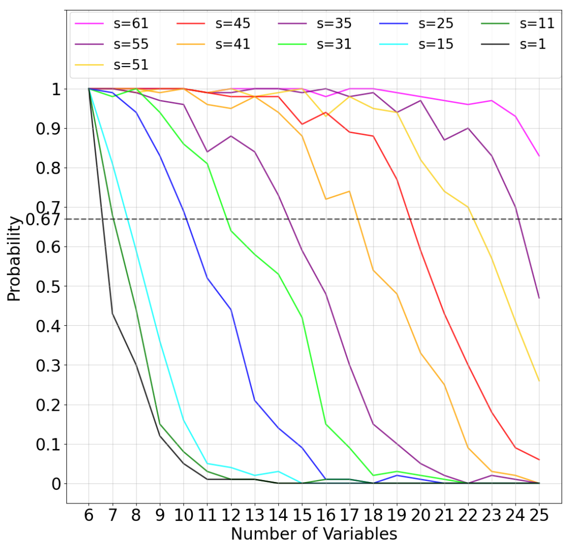
In Figure 2, we compare the practical complexities of the algorithms Search-LPTWY,

[1] To compute the parity of P with MultParityCount, we find the parity of the output of MultParityCount.

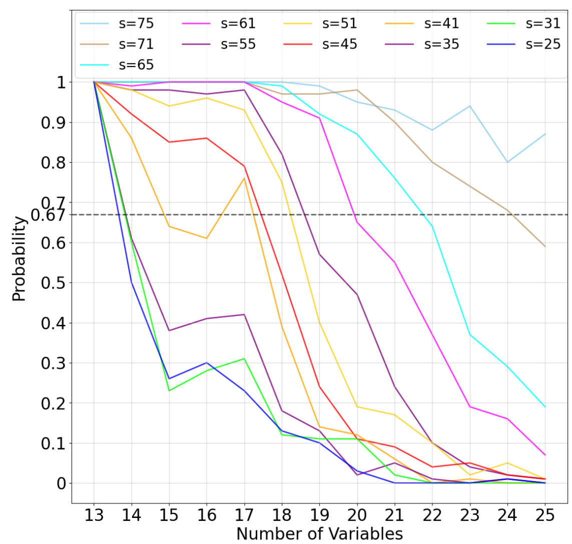
[2] Except in LPTWY with $n = 16, 17$ where we ran 20 iterations because each of them takes a considerable amount of time.



(a) Probability of success of Search-LPTWY with $\delta = 0.1235$ and several values of s.



(b) Probability of success of Search-BKW with $\lambda = 0.1967$ and several values of s.



(c) Probability of success of Search-Dinur1 with $\kappa = 0.31$, $\lambda = 0.2$, and several values of s.

Figure 1: Probabilities of success of Search-LPTWY, Search-BKW, and Search-Dinur1 for several values of s.

Search-BKW, Search-Dinur1, Dinur2, and Bruteforce. For each of them, we measure the number of clock cycles needed to solve a square system with a unique solution. For Search-LPTWY, Search-BKW, Search-Dinur1, we use the parameters suggested in the original papers with the exception of s, for which we used s_{min}, see Table 1. For Dinur2 we set $n_1 = \lfloor (1/5.4)n \rfloor$ satisfying the required $n_1 \approx (1/5.4)n$, so that Dinur2 has complexity $\mathcal{O}\left(n^2 2^{0.815n}\right)$ bit operations [12, Sec. 4]. For this configuration, we obtained that Dinur2 succeeds at finding a solution

Table 1: Estimation of s_{min}.

n	6	7	8	9	10	11	12	13	14	15	16	17
LPTWY	31	31	41	51	51	61	61	61	61	71	81	81
BKW	1	11	25	25	25	31	35	35	35	41	41	41
Dinur1	1	1	15	25	25	31	35	35	41	41	41	45
n	18	19	20	21	22	23	24	25				
BKW	45	45	51	51	51	55	55	61				
Dinur1	51	61	61	65	65	71	71	75				

Table 2: Growth rate of the practical complexity of solving a square quadratic system with at most one solution. In the case of Search-LPTWY, Search-BKW, Search-Dinur1, it means with probability of success greater than 2/3.

Algorithm	n_{max}	Experimental ($14 \leq n \leq n_{max}$)	Theoretical ($n \to \infty$)
Search-LPTWY	17	$2^{1.453}$	$2^{0.876}$
Search-BKW	25	$2^{0.876}$	$2^{0.804}$
Search-Dinur1	25	$2^{0.971}$	$2^{0.694}$
Dinur2	30	$2^{0.818}$	$2^{0.815}$
Bruteforce	30	$2^{1.022}$	2^{1}

with probability greater than 0.9 for every $6 \leq n \leq 20$.

From the data of Figure 2, we estimate the growth rates of the practical complexities of all probabilistic algorithms considered in this paper. The results are shown in Table 2. To have a more precise complexity estimation, we only use the data coming from $14 \leq n \leq n_{max}$, where n_{max} is the maximum value of n that we were able to test. We find that already for $n \geq 14$ Search-BKW, Search-Dinur1, and Dinur2 are catching up with brute-force, meaning that, as the number of variables increases by 1, the practical complexity of these algorithms increase by a factor less than 2, while the practical complexity of brute-force doubles. The same cannot be said for LPTWY, for which we see that in the range $n \in [14, 17]$ its practical complexity more than double as the number of variables increases by 1. Assuming (pessimistically) that the practical complexity of all the algorithms keep increasing by the factor shown in Table 2, we get that (our implementation of) brute-force is outperformed by Dinur1 for $n \geq 132$, BKW for $n \geq 60$, and by Dinur2 for $n \geq 33$. We remark that the growth rate of the practical complexity of Dinur2 is very close to its theoretical value as $n \to +\infty$, thus confirming Dinur's estimate.

We also measured the memory used by the algorithms Search-BKW, Search-Dinur1, and Dinur2 to solve random square quadratic systems for $n = 18, \ldots, 30$ (for $n \leq 17$, our implementation always uses about 2MB for default memory allocations), and compared it with the theoretical values of $s \cdot 2^{n-n_1}$, $s \cdot 2^{n-n_2}$, and $4n_1 \cdot 2^{n-n_1}$ bits, respectively. See Figure 3.

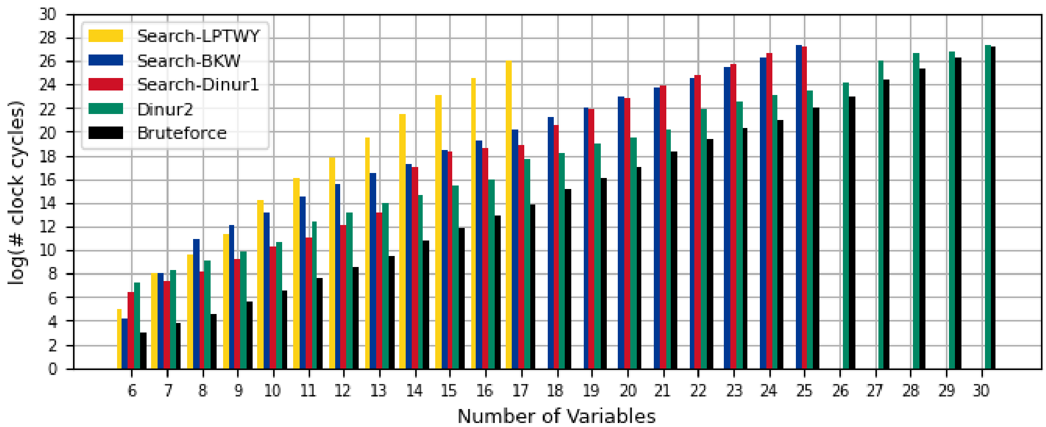


Figure 2: Clock-cycles of the probabilistic algorithms and Bruteforce on randomly generated square quadratic systems over $\mathbb{F}_2$ with a unique solution.

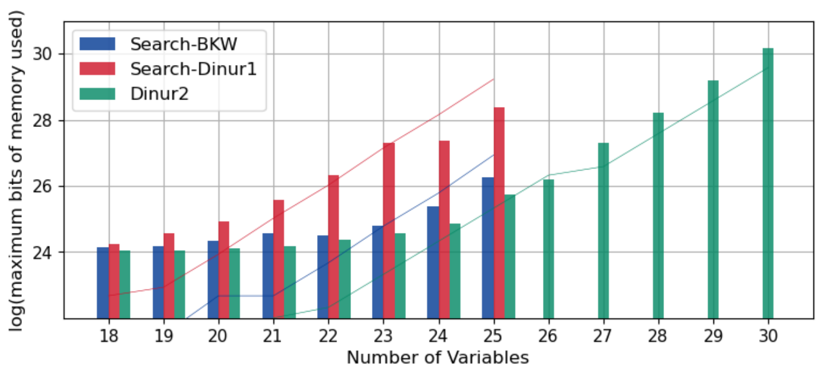


Figure 3: Memory usage of the probabilistic algorithms on randomly generated square quadratic systems over $\mathbb{F}_2$ (bars) compared with theoretical values (lines).

6 Conclusions and future works

Algorithms for solving Boolean polynomial systems are of essential importance in both pure and applied mathematics, and it is still unclear what is the best computational complexity that they can achieve. In fact, the computational complexities of many such algorithms are determined only under restrictive hypotheses and/or for the average case.

Lokshtanov et al. [22] were the first to exhibit an asymptotically lower time complexity than brute-force, by introducing a probabilistic algorithm that, in the worse case, solves a square polynomial system in time $O^*(2^{\delta n})$, for some $\delta \in (0, 1)$ depending on the degree of the system, without relying on any unproved hypothesis. Their result was improved by Björklund et al. [7] and Dinur [12, 13], who devised probabilistic algorithms with a smaller factor δ at the exponent.

In this paper, we survey the theory behind these probabilistic algorithms, which is based on the zeta and Möbius transforms and the Razborov–Smolensky construction, and which is much different from the approach of other algorithms for solving polynomial systems, e.g., Gröbner bases computation. We hope that by doing so we provided a useful resource for researchers interested in approaching these methods.

Moreover, motivated by the fact that the asymptotic complexity of an algorithm is not necessarily a good measure of its actual performance or real data, we illustrate the experimental results that we obtained by running our implementations of the probabilistic algorithms [29] on random square quadratic (Boolean) polynomial systems having exactly one solution.

First, we found that in the probabilistic algorithms the number s of iterations of the main loop can be significantly reduced, thus improving the speed by a factor of about 15, while keeping a reasonably high probability of success, see Figures 1(a), 1(b), 1(c), and Table 1.

Second, we compared the practical complexities (in terms of clock cycles) of the probabilistic algorithms, finding that for $n \geq 14$, despite being slower than brute-force, all algorithms except LPTWY are already catching up on brute-force, with respect to the growth rates of their practical complexities, see Figure 2 and Table 2. In particular, we found that already for $n \geq 14$ the growth rate of the practical complexity of Dinur2 is very close to its theoretical value as $n \to +\infty$, which confirms Dinur's results [12]. We estimate that in our implementation brute-force should be outperformed by BKW, Dinur1, and Dinur2 for $n \geq 60$, $n \geq 132$, and $n \geq 33$, respectively.

We believe that our study of the practical complexities shows that these probabilistic algorithms are not only very important theoretical results, but also have actual consequences for applications, that is, future efficient implementations of them could be among the fastest in solving Boolean polynomial systems. Indeed, although our current implementation of the probabilistic algorithms (which is written as a proof of concept, keeping it simple, and without focusing on optimizations) is slower than brute-force, it already shows that the growth rates of the practical complexities catch up quickly with their theoretical values, especially for Dinur2. Therefore, it is reasonable to expect that more efficient implementations of the probabilistic algorithms will beat brute-force already for small values of n. Efficient implementations of the zeta transform, for example exploiting parallelism via GPUs [6] or FPGAs, might be relevant. We hope that this work will contribute to stimulate further research in the direction of efficient implementations of these probabilistic algorithms and their potential descendants or variations.

References

[1] M. Bardet, J.-C. Faugére, and B. Salvy, *On the complexity of Gröbner basis computation of semi-regular overdetermined algebraic equations*, Proceedings of International Conference on Polynomial System Solving (ICPSS, Paris, France), 2004, pp. 71–75.

[2] M. Bardet, J.-C. Faugére, B. Salvy, and P.-J. Spaenlehauer, *On the complexity of solving quadratic Boolean systems*, Journal of Complexity **29** (2013), no. 1, 53–75.

[3] M. Bardet, J.-C. Faugére, B. Salvy, and B.-Y. Yang, *Asymptotic behaviour of the degree of regularity of semi-regular polynomial systems*, Proc. of MEGA 2005, Eighth International Symposium on Effective Methods in Algebraic Geometry, 2005.

[4] M. Beeler, R. W Gosper, and R. Schroeppel, *HAKMEM*, Tech. report, Massachusetts Institute of Technology, USA, 1972.

[5] L. Bettale, J.-C. Faugére, and L. Perret, *Hybrid approach for solving multivariate systems over finite fields*, Journal of Mathematical Cryptology **3** (2009), no. 3, 177–197.

[6] D. Bikov and I. Bouyukliev, *Parallel fast Möbius (Reed–Muller) transform and its implementation with CUDA on GPUs*, Proceedings of the International Workshop on Parallel Symbolic Computation (New York, NY, USA), PASCO 2017, Association for Computing Machinery, 2017.

[7] A. Björklund, P. Kaski, and R. Williams, *Solving systems of polynomial equations over* GF(2) *by a parity-counting self-reduction*, ICALP 2019, Leibniz International Proceedings in Informatics (LIPIcs), 2019, pp. 26:1–26:13.

[8] C. Bouillaguet, H.-C. Chen, C.-M. Cheng, T. Chou, R. Niederhagen, A. Shamir, and B.-Y. Yang, *Fast exhaustive search for polynomial systems in* $\mathbb{F}_2$, International Workshop on Cryptographic Hardware and Embedded Systems, Springer, 2010, pp. 203–218.

[9] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, Proceedings of the nineteenth annual ACM symposium on Theory of computing, 1987, pp. 1–6.

[10] N. Courtois, A. Klimov, J. Patarin, and A. Shamir, *Efficient algorithms for solving overdefined systems of multivariate polynomial equations*, International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2000, pp. 392–407.

[11] A. Dickenstein and I. Z. Emiris (eds.), *Solving Polynomial Equations*, Algorithms and Computation in Mathematics, vol. 14, Springer-Verlag, Berlin, 2005, Foundations, algorithms, and applications.

[12] I. Dinur, *Cryptanalytic applications of the polynomial method for solving multivariate equation systems over* $GF(2)$, EUROCRYPT 2021 available at https://eprint.iacr.org/2021/578.pdf, 2021.

[13] ———, *Improved algorithms for solving polynomial systems over* GF(2) *by multiple parity-counting*, Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, 2021, pp. 2550–2564.

[14] J. D. Duarte, *On the complexity of the crossbred algorithm*, Cryptology ePrint Archive, Report 2020/1058, 2020, https://eprint.iacr.org/2020/1058.

[15] J.-C. Faugére, *A new efficient algorithm for computing Gröbner bases (F4)*, Journal of pure and applied algebra **139** (1999), no. 1-3, 61–88.

[16] ———, *A new efficient algorithm for computing Gröbner bases without reduction to zero (F5)*, Proceedings of the 2002 international symposium on Symbolic and algebraic computation, 2002, pp. 75–83.

[17] J.-C. Faugère, M. S. El Din, and P.-J. Spaenlehauer, *Computing loci of rank defects of linear matrices using Gröbner bases and applications to Cryptology*, Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation (New York, NY, USA), ISSAC '10, Association for Computing Machinery, 2010, p. 257–264.

[18] A. S. Fraenkel and Y. Yesha, *Complexity of Problems in Games, Graphs and Algebraic Equations*, Discrete Applied Mathematics **1** (1979), no. 1, 15–30.

[19] R. Impagliazzo and R. Paturi, *On the complexity of k-SAT*, Journal of Computer and System Sciences **62** (2001), no. 2, 367–375.

[20] A. Joux and V. Vitse, *A crossbred algorithm for solving Boolean polynomial systems*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **10737 LNCS** (2018), 3–21.

[21] A. Kipnis, J. Patarin, and L. Goubin, *Unbalanced oil and vinegar signature schemes*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **1592** (1999), 206–222.

[22] D. Lokshtanov, R. Paturi, S. Tamaki, R. Williams, and H. Yu., *Beating brute force for systems of polynomial equations over finite fields.*, Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, 2017, pp. 2190–2202.

[23] Y. V. Matiyasevich, *Hilbert's tenth problem*, Foundations of Computing Series, MIT Press, Cambridge, MA, 1993, Translated from the 1993 Russian original by the author, With a foreword by Martin Davis.

[24] E. Mayr, *Membership in polynomial ideals over $\mathbb{Q}$ is exponential space complete*, STACS 89 (Berlin, Heidelberg) (B. Monien and R. Cori, eds.), Springer Berlin Heidelberg, 1989, pp. 400–406.

[25] H. Miura, Y. Hashimoto, and T. Takagi, *Extended algorithm for solving underdefined multivariate quadratic equations*, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences **E97-A** (2014), no. 6, 1418–1425.

[26] R. Niederhagen, K.-C. Ning, and B.-Y. Yang, *Implementing Joux-Vitse's crossbred algorithm for solving $\mathcal{MQ}$ systems over $\mathbb{F}_2$ on GPUs*, Post-Quantum Cryptography (Cham) (T. Lange and R. Steinwandt, eds.), Springer International Publishing, 2018, pp. 121–141.

[27] J. Patarin, *Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two new families of asymmetric algorithms*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **1070** (1996), 33–48.

[28] A. A. Razborov, *Lower bounds on the size of bounded-depth networks over a complete basis with logical addition*, Mathematical Notes of the Academy of Sciences of the USSR **41** (1987), 333–338.

[29] C. Sanna and J. Verbel, *Implementation of the probabilistic algorithms*, Available under Apache License 2.0, 2021, https://github.com/Crypto-TII/probabilistic_algorithms_for_boolean_polynomial_systems.

[30] R. Smolensky, *Algebraic methods in the theory of lower bounds for Boolean circuit complexity*, Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, pp. 77–82.

[31] V. Strassen, *Gaussian elimination is not optimal*, Numerische mathematik **13** (1969), no. 4, 354–356.

[32] B. Sturmfels, *Solving Systems of Polynomial Equations*, CBMS Regional Conference Series in Mathematics, vol. 97, Published for the Conference Board of the Mathematical Sciences, Washington, DC; by the American Mathematical Society, Providence, RI, 2002.

[33] L. G. Valiant and V. V. Vazirani, *NP is as easy as detecting unique solutions*, Theor. Comput. Sci. **47** (1986), 85–93.

[34] B.-Y. Yang and J.-M. Chen, *Theoretical analysis of XL over small fields*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **3108** (2004), 277–288.

[35] F. Yates, *The design and analysis of factorial experiments*, Tech. report, Imperial Bureau of Soil Science, Technical Communication no. 35, 1937.