Practical complexities of probabilistic algorithms for solving Boolean polynomial systems

January 19, 2022

Stefano Barbero [1], Emanuele Bellini [2], Carlo Sanna [1], and Javier Verbel [2]

[1] Politecnico di Torino, Torino, IT [2] Technology Innovation Institute, Abu Dhabi, UAE

Definition (Polynomial Solving Problem)

Input: a set of polynomials $f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n)$ in n unknowns with coefficients in $\mathbb{F}_q$

Output: $(a_1, \ldots, a_n) \in \mathbb{F}_q^n$ such that

$$f_1(a_1, \ldots, a_n) = \cdots = f_m(a_1, \ldots, a_n) = 0$$

Polynomial Solving in Cryptography

Polynomial Solving in Cryptography

Cryptography Research Centre

- $\mathcal{NP}$-complete, any decision problem reduces to it.

Polynomial Solving in Cryptography

- $\mathcal{NP}$-complete, any decision problem reduces to it.
- Directly to Multivariate cryptography

Polynomial Solving in Cryptography

- $\mathcal{NP}$-complete, any decision problem reduces to it.
- Directly to Multivariate cryptography
- Algebraic attacks:
  1. Legendre Pseudorandom Generation
  2. hash functions
  3. Cipher
  4. etc

Some algorithms to solve the polynomilas over finite fields

Some algorithms to solve the polynomilas over finite fields

Cryptography Research Centre

- Bruteforce (*complexity* $4 \log n \cdot 2^n$)

Some algorithms to solve the polynomilas over finite fields

Cryptography Research Centre

- Bruteforce (*complexity* $4 \log n \cdot 2^n$)
- direct linearization

Some algorithms to solve the polynomilas over finite fields

- Bruteforce (*complexity* $4 \log n \cdot 2^n$)
- direct linearization
- Extended linearization (XL)

Some algorithms to solve the polynomilas over finite fields

- Bruteforce (*complexity* $4 \log n \cdot 2^n$)
- direct linearization
- Extended linearization (XL)
- **Gröbner basis**: Buchberger's, F4, and F5.

Some algorithms to solve the polynomilas over finite fields

Cryptography Research Centre

- Bruteforce (*complexity* $4 \log n \cdot 2^n$)
- direct linearization
- Extended linearization (XL)
- **Gröbner basis**: Buchberger's, F4, and F5.
- **Hybrid approaches:** BooleanSolve, Hybrid-F5, Crossbred.

Some algorithms to solve the polynomilas over finite fields

- Bruteforce (*complexity* $4 \log n \cdot 2^n$)
- direct linearization
- Extended linearization (XL)
- **Gröbner basis**: Buchberger's, F4, and F5.
- **Hybrid approaches:** BooleanSolve, Hybrid-F5, Crossbred.
- special algorithms for underdefined systems ($m < n$)

Some algorithms to solve the polynomilas over finite fields

- Bruteforce (*complexity* $4 \log n \cdot 2^n$)
- direct linearization
- Extended linearization (XL)
- **Gröbner basis**: Buchberger's, F4, and F5.
- **Hybrid approaches:** BooleanSolve, Hybrid-F5, Crossbred.
- special algorithms for underdefined systems ($m < n$)

None of them outperform bruteforce asymptotically, in the worst-case!

Some algorithms to solve the polynomilas over finite fields

Cryptography Research Centre

- Bruteforce (*complexity* $4 \log n \cdot 2^n$)
- direct linearization
- Extended linearization (XL)
- **Gröbner basis**: Buchberger's, F4, and F5.
- **Hybrid approaches:** BooleanSolve, Hybrid-F5, Crossbred.
- special algorithms for underdefined systems ($m < n$)

None of them outperform bruteforce asymptotically, in the worst-case!

- (2017) **First algorithm asymptotically faster than bruteforce in the worst-case.**

  "*Beating Brute Force for Systems of Polynomial Equations over Finite Fields*" D. Lokshtanov, R. Patur, S. Tamaki, and R. Williams

Some algorithms to solve the polynomilas over finite fields

Cryptography Research Centre

- Bruteforce (*complexity* $4\log n \cdot 2^n$)
- direct linearization
- Extended linearization (XL)
- **Gröbner basis**: Buchberger's, F4, and F5.
- **Hybrid approaches:** BooleanSolve, Hybrid-F5, Crossbred.
- special algorithms for underdefined systems ($m < n$)

None of them outperform bruteforce asymptotically, in the worst-case!

- (2017) **First algorithm asymptotically faster than bruteforce in the worst-case.**

  "*Beating Brute Force for Systems of Polynomial Equations over Finite Fields*" D. Lokshtanov, R. Patur, S. Tamaki, and R. Williams

- (2019) Improved by A. Björklund, P. Kaski, and R. Williams

  Solving Systems of Polynomial Equations over GF(2) by a Parity-Counting Self-Reduction

Some algorithms to solve the polynomilas over finite fields

Cryptography Research Centre

- Bruteforce (*complexity* $4\log n \cdot 2^n$)
- direct linearization
- Extended linearization (XL)
- **Gröbner basis**: Buchberger's, F4, and F5.
- **Hybrid approaches:** BooleanSolve, Hybrid-F5, Crossbred.
- special algorithms for underdefined systems ($m < n$)

None of them outperform bruteforce asymptotically, in the worst-case!

- (2017) **First algorithm asymptotically faster than bruteforce in the worst-case.**

  "*Beating Brute Force for Systems of Polynomial Equations over Finite Fields*" D. Lokshtanov, R. Patur, S. Tamaki, and R. Williams

- (2019) Improved by A. Björklund, P. Kaski, and R. Williams

  Solving Systems of Polynomial Equations over GF(2) by a Parity-Counting Self-Reduction

- (2021) Improved by I. Dinur

  "*Solving Polynomial Systems over GF(2) by Multiple Parity-Counting*"
  "*Cryptanalytic Applications of the Polynomial Method for Solving Multivariate Equation Systems over GF(2)*"

Contents

Preliminaries concepts

Boolean function

Cryptography
Research
Centre

Boolean function

Definition: A *Boolean function* is a map $f : \mathbb{F}_2^n \to \mathbb{F}_2$.

Note: f can be uniquely represented in $\mathbb{F}_2[x_1, \ldots, x_n] / (x_1^2 - x_1, \ldots, x_n^2 - x_n)$.

$$f := \sum_{\mathbf{a} \in \mathbb{F}_2^n} \zeta_f(\mathbf{a}) \cdot \mathbf{x}^{\mathbf{a}}, \quad \text{where } \mathbf{x}^{\mathbf{a}} := x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n} \text{ and } \zeta_f(\mathbf{a}) \in \mathbb{F}_2$$

Boolean function

Cryptography Research Centre

Definition: A *Boolean function* is a map $f : \mathbb{F}_2^n \to \mathbb{F}_2$.

Note: f can be uniquely represented in $\mathbb{F}_2[x_1, \ldots, x_n] / (x_1^2 - x_1, \ldots, x_n^2 - x_n)$.

$$f := \sum_{\mathbf{a} \in \mathbb{F}_2^n} \zeta_f(\mathbf{a}) \cdot \mathbf{x}^{\mathbf{a}}, \quad \text{where } \mathbf{x}^{\mathbf{a}} := x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n} \text{ and } \zeta_f(\mathbf{a}) \in \mathbb{F}_2$$

Representing f as a vector of size 2^n.

$$\underbrace{[\zeta(\mathbf{a}) \mid \mathbf{a} \in \mathbb{F}_2^n]}_{\text{Algebraic Normal Form (ANF)}} \qquad \underbrace{[f(\mathbf{a}) \mid \mathbf{a} \in \mathbb{F}_2^n]}_{\text{Truth table}}$$

Boolean function

Cryptography Research Centre

Definition: A *Boolean function* is a map $f : \mathbb{F}_2^n \to \mathbb{F}_2$.

Note: f can be uniquely represented in $\mathbb{F}_2[x_1, \ldots, x_n] / (x_1^2 - x_1, \ldots, x_n^2 - x_n)$.

$$f := \sum_{\mathbf{a} \in \mathbb{F}_2^n} \zeta_f(\mathbf{a}) \cdot \mathbf{x}^{\mathbf{a}}, \qquad \text{where } \mathbf{x}^{\mathbf{a}} := x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n} \text{ and } \zeta_f(\mathbf{a}) \in \mathbb{F}_2$$

Representing f as a vector of size 2^n.

$$\underbrace{[\zeta(\mathbf{a}) \mid \mathbf{a} \in \mathbb{F}_2^n]}_{\text{Algebraic Normal Form (ANF)}} \quad \xleftrightarrow[\textit{Zeta transform}]{\zeta[f]} \quad \underbrace{[f(\mathbf{a}) \mid \mathbf{a} \in \mathbb{F}_2^n]}_{\text{Truth table}}$$

Boolean function

Definition: A *Boolean function* is a map $f : \mathbb{F}_2^n \to \mathbb{F}_2$.

Note: f can be uniquely represented in $\mathbb{F}_2[x_1, \ldots, x_n] / (x_1^2 - x_1, \ldots, x_n^2 - x_n)$.

$$f := \sum_{\mathbf{a} \in \mathbb{F}_2^n} \zeta_f(\mathbf{a}) \cdot \mathbf{x}^{\mathbf{a}}, \quad \text{where } \mathbf{x}^{\mathbf{a}} := x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n} \text{ and } \zeta_f(\mathbf{a}) \in \mathbb{F}_2$$

Representing f as a vector of size 2^n.

$$\underbrace{[\zeta(\mathbf{a}) \mid \mathbf{a} \in \mathbb{F}_2^n]}_{\text{Algebraic Normal Form (ANF)}} \xleftrightarrow[\textit{Zeta transform}]{\zeta[f]} \underbrace{[f(\mathbf{a}) \mid \mathbf{a} \in \mathbb{F}_2^n]}_{\text{Truth table}}$$

- $\zeta[\text{ANF of } f] = \text{truth table of } f, \quad \zeta[\text{truth table of } f] = \text{ANF of } f$
- $\zeta[\zeta[f]] = f$
- *Complexity* $= O(n2^n)$

Interpolation of Boolean function

[1] **downward closed set**: if $\mathcal{A} \subseteq \mathbb{F}_2^n$ is a *downward closed set*, that is, if $\mathbf{a} \in \mathcal{A}$ implies that $\mathbf{b} \in \mathcal{A}$ for every $\mathbf{b} \in \mathbb{F}_2^n$ with $\mathbf{b} \leq \mathbf{a}$

Interpolation of Boolean function

Cryptography Research Centre

Interpolation algorithm

Input : The **partial** truth table $[f(\mathbf{a}) : \mathbf{a} \in \mathcal{A}]$ with $\mathrm{supp}(f) \subseteq \mathcal{A}$, and $\mathcal{A} \subseteq \mathbb{F}_2^n$ a downward closed set. [1]

Output: The **whole** truth table $[f(\mathbf{a}) : \mathbf{a} \in \mathbb{F}_2^n]$.

[1] **downward closed set**: if $\mathcal{A} \subseteq \mathbb{F}_2^n$ is a *downward closed set*, that is, if $\mathbf{a} \in \mathcal{A}$ implies that $\mathbf{b} \in \mathcal{A}$ for every $\mathbf{b} \in \mathbb{F}_2^n$ with $\mathbf{b} \leq \mathbf{a}$

Interpolation of Boolean function

Cryptography Research Centre

Interpolation algorithm

Input : The **partial** truth table $[f(\mathbf{a}) : \mathbf{a} \in \mathcal{A}]$ with $\mathrm{supp}(f) \subseteq \mathcal{A}$, and $\mathcal{A} \subseteq \mathbb{F}_2^n$ a downward closed set. [1]

Output: The **whole** truth table $[f(\mathbf{a}) : \mathbf{a} \in \mathbb{F}_2^n]$.

- it has complexity $O(n2^n)$.

[1] **downward closed set**: if $\mathcal{A} \subseteq \mathbb{F}_2^n$ is a *downward closed set*, that is, if $\mathbf{a} \in \mathcal{A}$ implies that $\mathbf{b} \in \mathcal{A}$ for every $\mathbf{b} \in \mathbb{F}_2^n$ with $\mathbf{b} \leq \mathbf{a}$

Interpolation of Boolean function

Interpolation algorithm

Input : The **partial** truth table $[f(\mathbf{a}) : \mathbf{a} \in \mathcal{A}]$ with supp$(f) \subseteq \mathcal{A}$, and $\mathcal{A} \subseteq \mathbb{F}_2^n$ a downward closed set. [1]

Output: The **whole** truth table $[f(\mathbf{a}) : \mathbf{a} \in \mathbb{F}_2^n]$.

- it has complexity $O(n2^n)$.
- if f has degree d, then know $[f(\mathbf{a}) : \mathbf{a} \in \mathbb{F}_2^n,\ \text{ and } wt(\mathbf{a}) \leq d]$ has enough information to compute the **whole** truth table of f

[1] **downward closed set**: if $\mathcal{A} \subseteq \mathbb{F}_2^n$ is a *downward closed set*, that is, if $\mathbf{a} \in \mathcal{A}$ implies that $\mathbf{b} \in \mathcal{A}$ for every $\mathbf{b} \in \mathbb{F}_2^n$ with $\mathbf{b} \leq \mathbf{a}$

Characteristic polynomial of a system

Characteristic polynomial of a system

Cryptography Research Centre

Definition (Characteristic polynomial of system of polynomials)

The polynomial

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

is called the *characteristic polynomial* of the system $p_1(\mathbf{x}), \ldots, p_m(\mathbf{x})$.

Characteristic polynomial of a system

Cryptography Research Centre

Definition (Characteristic polynomial of system of polynomials)

The polynomial

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

is called the *characteristic polynomial* of the system $p_1(\mathbf{x}), \ldots, p_m(\mathbf{x})$.

Properties

- $F(\mathbf{a}) = 1 \Leftrightarrow p_1(\mathbf{a}) = \cdots p_m(\mathbf{a}) = 0$
- $\deg(p_i) = d \implies \deg(F) = md$ (very high)

Characteristic polynomial of a system

Definition (Characteristic polynomial of system of polynomials)

The polynomial

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

is called the *characteristic polynomial* of the system $p_1(\mathbf{x}), \ldots, p_m(\mathbf{x})$.

Properties

- $F(\mathbf{a}) = 1 \Leftrightarrow p_1(\mathbf{a}) = \cdots p_m(\mathbf{a}) = 0$
- $\deg(p_i) = d \implies \deg(F) = md$ (very high)
- $G : \mathbb{F}_2^{n-n_1} \to \mathbb{F}_2$ s.t. $G(\mathbf{y}) := \sum_{\mathbf{c} \in \mathbb{F}_2^{n_1}} F(\mathbf{y}, \mathbf{c})$

  $G(\mathbf{b}) = 1 \implies F(\mathbf{b}, \mathbf{c}) = 1$ for some $\mathbf{c} \in \mathbb{F}_2^{n_1}$.

$$\underline{G : \mathbb{F}_2^{n-n_1}} \to \mathbb{F}_2 \quad \textbf{s.t.} \quad G(\mathbf{y}) := \sum_{\mathbf{c} \in \mathbb{F}_2^{n_1}} F(\mathbf{y}, \mathbf{c})$$

$\underline{G : \mathbb{F}_2^{n-n_1}} \to \mathbb{F}_2$ **s.t.** $G(\mathbf{y}) := \sum_{\mathbf{c} \in \mathbb{F}_2^{n_1}} F(\mathbf{y}, \mathbf{c})$

1. $G(\mathbf{b}) = 1 \implies F(\mathbf{b}, \mathbf{c}) = 1$ for some $\mathbf{c} \in \mathbb{F}_2^{n_1}$.
2. $\deg(G) = dm - n_1$ (Still very high!).

$G : \mathbb{F}_2^{n-n_1} \to \mathbb{F}_2$ **s.t.** $G(\mathbf{y}) := \sum_{\mathbf{c}\in\mathbb{F}_2^{n_1}} F(\mathbf{y},\mathbf{c})$

Cryptography Research Centre

1. $G(\mathbf{b}) = 1 \implies F(\mathbf{b},\mathbf{c}) = 1$ for some $\mathbf{c} \in \mathbb{F}_2^{n_1}$.
2. $\deg(G) = dm - n_1$ (Still very high!).
3. Parity $:= \sum_{\mathbf{b}\in\mathbb{F}_2^{n-n_1}} G(\mathbf{b}) = \sum_{\mathbf{a}\in\mathbb{F}_2^{n}} F(\mathbf{a})$ (parity of the number of solutions)

$G : \mathbb{F}_2^{n-n_1} \to \mathbb{F}_2$ **s.t.** $G(\mathbf{y}) := \sum_{\mathbf{c}\in\mathbb{F}_2^{n_1}} F(\mathbf{y},\mathbf{c})$

Cryptography Research Centre

1. $G(\mathbf{b}) = 1 \implies F(\mathbf{b},\mathbf{c}) = 1$ for some $\mathbf{c} \in \mathbb{F}_2^{n_1}$.
2. $\deg(G) = dm - n_1$ (Still very high!).
3. Parity $:= \sum_{\mathbf{b}\in\mathbb{F}_2^{n-n_1}} G(\mathbf{b}) = \sum_{\mathbf{a}\in\mathbb{F}_2^{n}} F(\mathbf{a})$ (parity of the number of solutions)

Some algorithms computes a poly $\tilde{G}$ approximating G

- **Björklund et al.'s:** (many $\tilde{G}$) to compute Parity.
- **Dinur's first:** (many $\tilde{G}$) to compute Parity.
- **Dinur's second:** (few $\tilde{G}$) A method to estimate $\mathbf{c}$ for every $\mathbf{b}$ such that $\tilde{G}(\mathbf{b}) = 1$.

$G : \mathbb{F}_2^{n-n_1} \to \mathbb{F}_2$ **s.t.** $G(\mathbf{y}) := \sum_{\mathbf{c}\in\mathbb{F}_2^{n_1}} F(\mathbf{y}, \mathbf{c})$

Cryptography Research Centre

1. $G(\mathbf{b}) = 1 \implies F(\mathbf{b}, \mathbf{c}) = 1$ for some $\mathbf{c} \in \mathbb{F}_2^{n_1}$.
2. $\deg(G) = dm - n_1$ (Still very high!).
3. Parity $:= \sum_{\mathbf{b}\in\mathbb{F}_2^{n-n_1}} G(\mathbf{b}) = \sum_{\mathbf{a}\in\mathbb{F}_2^{n}} F(\mathbf{a})$ (parity of the number of solutions)

Some algorithms computes a poly $\tilde{G}$ approximating G

- **Björklund et al.'s:** (many $\tilde{G}$) to compute Parity.
- **Dinur's first:** (many $\tilde{G}$) to compute Parity.
- **Dinur's second:** (few $\tilde{G}$) A method to estimate $\mathbf{c}$ for every $\mathbf{b}$ such that $\tilde{G}(\mathbf{b}) = 1$.

Lokshtanov et al.'s: Determine the **consistency** by a precise approx. of V, where

$$V(\mathbf{b}) = \sum_{\mathbf{c}\in\mathbb{F}_2^{n_1}} s_{\mathbf{c}} F(\mathbf{b}, \mathbf{c})$$

Approximation techniques

If $\Pr\left[\tilde{F}(\mathbf{a}) = F(\mathbf{a})\right]$ is close to one, then $\tilde{F}$ approximates F.

The Razborov–Smolensky construction

Let $\ell \in \{1, \ldots, m\}$ be an integer. Define

$$F(\mathbf{x}) = \prod_{i=1}^{m}(1 + p_i(\mathbf{x})) \quad \tilde{F}(x) := \prod_{i=1}^{\ell}(1 + R_i(\mathbf{x})),$$

where $R_i(\mathbf{x}) := \sum_{j=1}^{m} \alpha_{ij} p_j(\mathbf{x})$, and the $\alpha_{ij} \in \mathbb{F}_2$ are chosen uniformly at random.

Approximation techniques

If $\Pr\left[\tilde{F}(\mathbf{a}) = F(\mathbf{a})\right]$ is close to one, then $\tilde{F}$ approximates F.

The Razborov–Smolensky construction

Let $\ell \in \{1, \dots, m\}$ be an integer. Define

$$F(\mathbf{x}) = \prod_{i=1}^{m}(1 + p_i(\mathbf{x})) \quad \tilde{F}(x) := \prod_{i=1}^{\ell}(1 + R_i(\mathbf{x})),$$

where $R_i(\mathbf{x}) := \sum_{j=1}^{m} \alpha_{ij} p_j(\mathbf{x})$, and the $\alpha_{ij} \in \mathbb{F}_2$ are chosen uniformly at random.

- $\deg(\tilde{F}) \leq d\ell$

Approximation techniques

Cryptography Research Centre

If $\Pr\left[\tilde{F}(\mathbf{a}) = F(\mathbf{a})\right]$ is close to one, then $\tilde{F}$ approximates F.

The Razborov–Smolensky construction

Let $\ell \in \{1, \ldots, m\}$ be an integer. Define

$$F(\mathbf{x}) = \prod_{i=1}^{m}(1 + p_i(\mathbf{x})) \quad \tilde{F}(x) := \prod_{i=1}^{\ell}(1 + R_i(\mathbf{x})),$$

where $R_i(\mathbf{x}) := \sum_{j=1}^{m} \alpha_{ij} p_j(\mathbf{x})$, and the $\alpha_{ij} \in \mathbb{F}_2$ are chosen uniformly at random.

- $\deg(\tilde{F}) \leq d\ell$
- $F(\mathbf{a}) = 1 \to \tilde{F}(\mathbf{a}) = 1$, otherwise

Approximation techniques

If $\Pr\left[\tilde{F}(\mathbf{a}) = F(\mathbf{a})\right]$ is close to one, then $\tilde{F}$ approximates F.

The Razborov–Smolensky construction

Let $\ell \in \{1, \ldots, m\}$ be an integer. Define

$$F(\mathbf{x}) = \prod_{i=1}^{m}(1 + p_i(\mathbf{x})) \quad \tilde{F}(x) := \prod_{i=1}^{\ell}(1 + R_i(\mathbf{x})),$$

where $R_i(\mathbf{x}) := \sum_{j=1}^{m} \alpha_{ij} p_j(\mathbf{x})$, and the $\alpha_{ij} \in \mathbb{F}_2$ are chosen uniformly at random.

- $\deg(\tilde{F}) \leq d\ell$
- $F(\mathbf{a}) = 1 \to \tilde{F}(\mathbf{a}) = 1$, otherwise
- $\Pr\left[\tilde{F}(\mathbf{a}) = F(\mathbf{a})\right] \geq 1 - 2^{-\ell}$

Approximation techniques

If $\Pr\left[\tilde{F}(\mathbf{a}) = F(\mathbf{a})\right]$ is close to one, then $\tilde{F}$ approximates F.

The Razborov–Smolensky construction

Let $\ell \in \{1, \ldots, m\}$ be an integer. Define

$$F(\mathbf{x}) = \prod_{i=1}^{m}(1 + p_i(\mathbf{x})) \quad \tilde{F}(x) := \prod_{i=1}^{\ell}(1 + R_i(\mathbf{x})),$$

where $R_i(\mathbf{x}) := \sum_{j=1}^{m} \alpha_{ij} p_j(\mathbf{x})$, and the $\alpha_{ij} \in \mathbb{F}_2$ are chosen uniformly at random.

- $\deg(\tilde{F}) \leq d\ell$
- $F(\mathbf{a}) = 1 \to \tilde{F}(\mathbf{a}) = 1$, otherwise
- $\Pr\left[\tilde{F}(\mathbf{a}) = F(\mathbf{a})\right] \geq 1 - 2^{-\ell}$
- $\tilde{G}(\mathbf{y}) := \sum_{\mathbf{c} \in \mathbb{F}_2^{n_1}} \tilde{F}(\mathbf{y}, \mathbf{c})$

Approximation techniques

If $\Pr\left[\tilde{F}(\mathbf{a}) = F(\mathbf{a})\right]$ is close to one, then $\tilde{F}$ approximates F.

The Razborov–Smolensky construction

Let $\ell \in \{1, \ldots, m\}$ be an integer. Define

$$F(\mathbf{x}) = \prod_{i=1}^{m}(1 + p_i(\mathbf{x})) \quad \tilde{F}(x) := \prod_{i=1}^{\ell}(1 + R_i(\mathbf{x})),$$

where $R_i(\mathbf{x}) := \sum_{j=1}^{m} \alpha_{ij} p_j(\mathbf{x})$, and the $\alpha_{ij} \in \mathbb{F}_2$ are chosen uniformly at random.

- $\deg(\tilde{F}) \leq d\ell$
- $F(\mathbf{a}) = 1 \to \tilde{F}(\mathbf{a}) = 1$, otherwise
- $\Pr\left[\tilde{F}(\mathbf{a}) = F(\mathbf{a})\right] \geq 1 - 2^{-\ell}$
- $\tilde{G}(\mathbf{y}) := \sum_{\mathbf{c} \in \mathbb{F}_2^{n_1}} \tilde{F}(\mathbf{y}, \mathbf{c})$

- $\tilde{V}(\mathbf{b}) := \sum_{\mathbf{c} \in \mathbb{F}_2^{n_1}} s_{\mathbf{c}} \tilde{F}(\mathbf{b}, \mathbf{c})$

Approximation techniques

If $\Pr\left[\tilde{F}(\mathbf{a}) = F(\mathbf{a})\right]$ is close to one, then $\tilde{F}$ approximates F.

The Razborov–Smolensky construction

Let $\ell \in \{1, \ldots, m\}$ be an integer. Define

$$F(\mathbf{x}) = \prod_{i=1}^{m}(1 + p_i(\mathbf{x})) \quad \tilde{F}(x) := \prod_{i=1}^{\ell}(1 + R_i(\mathbf{x})),$$

where $R_i(\mathbf{x}) := \sum_{j=1}^{m} \alpha_{ij} p_j(\mathbf{x})$, and the $\alpha_{ij} \in \mathbb{F}_2$ are chosen uniformly at random.

- $\deg(\tilde{F}) \leq d\ell$
- $F(\mathbf{a}) = 1 \rightarrow \tilde{F}(\mathbf{a}) = 1$, otherwise
- $\Pr\left[\tilde{F}(\mathbf{a}) = F(\mathbf{a})\right] \geq 1 - 2^{-\ell}$
- $\tilde{G}(\mathbf{y}) := \sum_{\mathbf{c} \in \mathbb{F}_2^{n_1}} \tilde{F}(\mathbf{y}, \mathbf{c})$

- $\tilde{V}(\mathbf{b}) := \sum_{\mathbf{c} \in \mathbb{F}_2^{n_1}} s_{\mathbf{c}} \tilde{F}(\mathbf{b}, \mathbf{c})$
- Still, erros appear for many $\mathbf{b} \in \mathbb{F}_2^{n-n_1}$

Approximation techniques

Cryptography Research Centre

If $\Pr\left[\tilde{F}(\mathbf{a}) = F(\mathbf{a})\right]$ is close to one, then $\tilde{F}$ approximates F.

The Razborov–Smolensky construction

Let $\ell \in \{1, \dots, m\}$ be an integer. Define

$$F(\mathbf{x}) = \prod_{i=1}^{m}(1 + p_i(\mathbf{x})) \quad \tilde{F}(x) := \prod_{i=1}^{\ell}(1 + R_i(\mathbf{x})),$$

where $R_i(\mathbf{x}) := \sum_{j=1}^{m} \alpha_{ij} p_j(\mathbf{x})$, and the $\alpha_{ij} \in \mathbb{F}_2$ are chosen uniformly at random.

- $\deg(\tilde{F}) \leq d\ell$
- $F(\mathbf{a}) = 1 \rightarrow \tilde{F}(\mathbf{a}) = 1$, otherwise
- $\Pr\left[\tilde{F}(\mathbf{a}) = F(\mathbf{a})\right] \geq 1 - 2^{-\ell}$
- $\tilde{G}(\mathbf{y}) := \sum_{\mathbf{c} \in \mathbb{F}_2^{n_1}} \tilde{F}(\mathbf{y}, \mathbf{c})$

- $\tilde{V}(\mathbf{b}) := \sum_{\mathbf{c} \in \mathbb{F}_2^{n_1}} s_{\mathbf{c}} \tilde{F}(\mathbf{b}, \mathbf{c})$
- Still, erros appear for many $\mathbf{b} \in \mathbb{F}_2^{n-n_1}$
- generate many $\tilde{F}$ and define $G(\mathbf{b}) = 1 \iff \#\{\tilde{F} : \tilde{G}(\mathbf{b}) = 1\} > t_0$, for a fixed integer t_0.

Probabilistic algorithms

Lokshtanov et al.'s

Suppose that the input polynomials have degree d.

Lokshtanov et al.'s

Suppose that the input polynomials have degree d.

- parameters: n, n_1, s, ℓ
- for $t = 1, \ldots, s$

1. Symbolically compute the ANF

$$\tilde{V}_0(\mathbf{y}) = \sum_{\mathbf{c} \in \mathbb{F}_2^{n_1}} s_{\mathbf{c}} \tilde{F}(\mathbf{y}, \mathbf{c}),$$

   with a new $\tilde{F}$ for each $\mathbf{c}$.

2. compute $\zeta(V_0)$ (truth table of V_0) .

Lokshtanov et al.'s

Suppose that the input polynomials have degree d.

- parameters: n, n_1, s, ℓ
- for $t = 1, \dots, s$
  1. Symbolically compute the ANF

$$\tilde{V}_0(\mathbf{y}) = \sum_{\mathbf{c}\in\mathbb{F}_2^{n_1}} s_{\mathbf{c}} \tilde{F}(\mathbf{y}, \mathbf{c}),$$

     with a new $\tilde{F}$ for each $\mathbf{c}$.
  2. compute $\zeta(V_0)$ (truth table of V_0) .
  3. store $\zeta(V_0)$

Lokshtanov et al.'s

Suppose that the input polynomials have degree d.

- parameters: n, n_1, s, ℓ
- for $t = 1, \dots, s$
  1. Symbolically compute the ANF

     $$\tilde{V}_0(\mathbf{y}) = \sum_{\mathbf{c} \in \mathbb{F}_2^{n_1}} s_{\mathbf{c}} \tilde{F}(\mathbf{y}, \mathbf{c}),$$

     with a new $\tilde{F}$ for each $\mathbf{c}$.
  2. compute $\zeta(V_0)$ (truth table of V_0) .
  3. store $\zeta(V_0)$
- for $\mathbf{b} \in \mathbb{F}_2^{n-n_1}$:
  if $\#\{\tilde{V}(\mathbf{b}) = 1\} > 0.4s$:
  return True.
- otherwise **return** False

Lokshtanov et al.'s

Suppose that the input polynomials have degree d.

- parameters: n, n_1, s, ℓ
- for $t = 1, \ldots, s$
  1. Symbolically compute the ANF
  $$\tilde{V}_0(\mathbf{y}) = \sum_{\mathbf{c} \in \mathbb{F}_2^{n_1}} s_{\mathbf{c}} \tilde{F}(\mathbf{y}, \mathbf{c}),$$
  with a new $\tilde{F}$ for each $\mathbf{c}$.
  2. compute $\zeta(V_0)$ (truth table of V_0) .
  3. store $\zeta(V_0)$
- for $\mathbf{b} \in \mathbb{F}_2^{n-n_1}$:
  if $\#\{\tilde{V}(\mathbf{b}) = 1\} > 0.4s$:
  return True.
- otherwise **return** False

Complexity

First loop: $T_1 = O^*\left(2^{n_1} \cdot \binom{n-n_1}{\downarrow d\ell - n_1}\right)$

Second loop: $T_2 = O^*\left(2^{n-n_1}\right)$

Setting $\ell = n_1 + 2$, $n_1 = \lfloor \delta n \rfloor$, and choosing δ s.t $T_1 \approx T_2$ we have

$$\textit{complexity} = \begin{cases} O^*\left(2^{0.8756n}\right) & \text{if } d = 2, \textit{and} \\ O^*\left(2^{(1-1/(5d))n}\right) & \text{if } d > 2 \end{cases}$$

Björklund et al.'s

Input polynomials $p_1, \ldots, p_m$ have degree d, and $\mathcal{W}_t^n := \{\mathbf{a} \in \mathbb{F}_2^n \mid wt(\mathbf{a}) \leq t\}$

Björklund et al.'s

Input polynomials $p_1, \ldots, p_m$ have degree d, and $\mathcal{W}_t^n := \{\mathbf{a} \in \mathbb{F}_2^n \mid wt(\mathbf{a}) \leq t\}$

- parameters: n, n_1, s, ℓ
  Computes $\left[G(\mathbf{b}) \;:\; \mathbf{b} \in \mathbb{F}_2^{n-n_1}\right]$

Björklund et al.'s

Cryptography Research Centre

Input polynomials $p_1, \ldots, p_m$ have degree d, and $\mathcal{W}_t^n := \{\mathbf{a} \in \mathbb{F}_2^n \mid wt(\mathbf{a}) \leq t\}$

- parameters: n, n_1, s, ℓ
  Computes $\left[G(\mathbf{b}) \ : \ \mathbf{b} \in \mathbb{F}_2^{n-n_1}\right]$
- for $k = 1, \ldots, s$
  1. $\tilde{G}(\mathbf{b})$ for each $\mathbf{b} \in \mathcal{W}_{d\ell-n_1}^{n-n_1}$ by recursive calls to Björklund's algo.

     Note: $\tilde{G}(\mathbf{b}) = \sum_{\mathbf{z} \in \mathbb{F}_2^{n_1}} \tilde{F}(\mathbf{b}, \mathbf{z})$.

Björklund et al.'s

Input polynomials $p_1, \ldots, p_m$ have degree d, and $\mathcal{W}_t^n := \{\mathbf{a} \in \mathbb{F}_2^n \mid wt(\mathbf{a}) \leq t\}$

- parameters: n, n_1, s, ℓ
  Computes $\left[G(\mathbf{b}) \;:\; \mathbf{b} \in \mathbb{F}_2^{n-n_1}\right]$
- for $k = 1, \ldots, s$
  1. $\tilde{G}(\mathbf{b})$ for each $\mathbf{b} \in \mathcal{W}_{d\ell-n_1}^{n-n_1}$ by recursive calls to Björklund's algo.

     Note: $\tilde{G}(\mathbf{b}) = \sum_{\mathbf{z} \in \mathbb{F}_2^{n_1}} \tilde{F}(\mathbf{b}, \mathbf{z})$.
  2. Interpolate $\tilde{G}$ and store it

Björklund et al.'s

Cryptography Research Centre

Input polynomials $p_1, \ldots, p_m$ have degree d, and $\mathcal{W}_t^n := \{\mathbf{a} \in \mathbb{F}_2^n \mid wt(\mathbf{a}) \le t\}$

- parameters: n, n_1, s, ℓ
  Computes $\left[G(\mathbf{b}) \ : \ \mathbf{b} \in \mathbb{F}_2^{n-n_1}\right]$
- for $k = 1, \ldots, s$
  1. $\tilde{G}(\mathbf{b})$ for each $\mathbf{b} \in \mathcal{W}_{d\ell-n_1}^{n-n_1}$ by recursive calls to Björklund's algo.

     Note: $\tilde{G}(\mathbf{b}) = \sum_{\mathbf{z} \in \mathbb{F}_2^{n_1}} \tilde{F}(\mathbf{b}, \mathbf{z})$.
  2. Interpolate $\tilde{G}$ and store it
- for $\mathbf{b} \in \mathbb{F}_2^{n-n_1}$:
  if $\#\{\tilde{F} \mid \tilde{G}(\mathbf{b}) = 1\} > s/2$:
  $G(\mathbf{b}) := 1$ (otherwise 0)

return $\sum_{\mathbf{b}} G(\mathbf{b})$

Björklund et al.'s

Input polynomials $p_1, \dots, p_m$ have degree d, and $\mathcal{W}_t^n := \{\mathbf{a} \in \mathbb{F}_2^n \mid wt(\mathbf{a}) \leq t\}$

- parameters: n, n_1, s, ℓ
  Computes $\left[G(\mathbf{b}) \;:\; \mathbf{b} \in \mathbb{F}_2^{n-n_1}\right]$
- for $k = 1, \dots, s$
  1. $\tilde{G}(\mathbf{b})$ for each $\mathbf{b} \in \mathcal{W}_{d\ell-n_1}^{n-n_1}$ by recursive calls to Björklund's algo.

     Note: $\tilde{G}(\mathbf{b}) = \sum_{\mathbf{z} \in \mathbb{F}_2^{n_1}} \tilde{F}(\mathbf{b}, \mathbf{z})$.
  2. Interpolate $\tilde{G}$ and store it
- for $\mathbf{b} \in \mathbb{F}_2^{n-n_1}$:
  if $\#\{\tilde{F} \mid \tilde{G}(\mathbf{b}) = 1\} > s/2$:
  $G(\mathbf{b}) := 1$ (otherwise 0)

return $\sum_{\mathbf{b}} G(\mathbf{b})$

Complexity

$T(n) =$ time of a size n instance

Rec. calls: $T_1 = O^*\left(T(n_1) \cdot \binom{n-n_1}{\downarrow d\ell - n_1}\right)$

Interpolation and last loop: $T_2 = O^*\left(2^{n-n_1}\right)$

Similarly, we force $T_1 \approx T_2$ so that have

$$\textit{complexity} = \begin{cases} O^*\left(2^{0.804n}\right) & \text{if } d = 2, \textit{and} \\ O^*\left(2^{(1-1/(2.7d))n}\right) & \text{if } d > 2 \end{cases}$$

Dinur's first

Similarly $\mathcal{W}_w^n := \{\mathbf{a} \in \mathbb{F}_2^n \mid wt(\mathbf{a}) \leq w\}$

- parameters: $n, n_1, {\color{red}n_2} < n_1, s, \ell,$

Dinur's first

Similarly $\mathcal{W}_w^n := \{\mathbf{a} \in \mathbb{F}_2^n \mid wt(\mathbf{a}) \leq w\}$

- parameters: $n, n_1, n_2 < n_1, s, \ell,$
- Compute $\left[G(\mathbf{b}) \ : \ \mathbf{b} \in \mathbb{F}_2^{n-n_1}\right]$ by **one** recursive call to the algorithm computing

$$[G(\mathbf{b}) \ : \ \mathbf{b} \in \mathcal{W}_w^{n-n_1}]$$

Dinur's first

Similarly $\mathcal{W}_w^n := \{\mathbf{a} \in \mathbb{F}_2^n \mid wt(\mathbf{a}) \leq w\}$

- parameters: $n, n_1, n_2 < n_1, s, \ell,$
- Compute $\left[G(\mathbf{b}) \;:\; \mathbf{b} \in \mathbb{F}_2^{n-n_1}\right]$ by **one** recursive call to the algorithm computing

$$[G(\mathbf{b}) \;:\; \mathbf{b} \in \mathcal{W}_w^{n-n_1}]$$

- Finally,

return Parity $= \sum_{\mathbf{b} \in \mathbb{F}_2^{n-n_1}} G(\mathbf{b})$

Dinur's first

Similarly $\mathcal{W}_w^n := \{\mathbf{a} \in \mathbb{F}_2^n \mid wt(\mathbf{a}) \leq w\}$

- parameters: $n, n_1, n_2 < n_1, s, \ell,$
- Compute $\left[G(\mathbf{b}) \;:\; \mathbf{b} \in \mathbb{F}_2^{n-n_1}\right]$ by **one** recursive call to the algorithm computing

$$\left[G(\mathbf{b}) \;:\; \mathbf{b} \in \mathcal{W}_w^{n-n_1}\right]$$

- Finally,

return Parity $= \sum_{\mathbf{b} \in \mathbb{F}_2^{n-n_1}} G(\mathbf{b})$

Complexity

$$\textit{complexity} = \begin{cases} O^*\left(2^{0.6943n}\right) \text{ if } d = 2, and \\ O^*\left(2^{(1-1/(2d))n}\right) \text{ if } d > 2 \end{cases}$$

Dinur's second

Let $p_1, \ldots, p_m$ be the input polys. Here we doesn't accurately compute G.

Dinur's second

Let $p_1, \ldots, p_m$ be the input polys. Here we doesn't accurately compute G.

parameters: n, n_1

Repeat few times do the following:

Dinur's second

Let $p_1, \ldots, p_m$ be the input polys. Here we doesn't accurately compute G.
parameters: n, n_1

Repeat few times do the following:

- Generate $R_1, \ldots, R_{n_1+1}$ random lin. comb. of $p_1, \ldots, p_m$

Dinur's second

Let $p_1, \ldots, p_m$ be the input polys. Here we doesn't accurately compute G.
parameters: n, n_1

Repeat few times do the following:

- Generate $R_1, \ldots, R_{n_1+1}$ random lin. comb. of $p_1, \ldots, p_m$
- compute the truth table of $\tilde{G}$

Dinur's second

Let $p_1, \ldots, p_m$ be the input polys. Here we doesn't accurately compute G.
parameters: n, n_1

Repeat few times do the following:

- Generate $R_1, \ldots, R_{n_1+1}$ random lin. comb. of $p_1, \ldots, p_m$
- compute the truth table of $\tilde{G}$

1. $\forall\ (\mathbf{b}, \mathbf{c}) \in \mathcal{W}^{n-n_1}_{d(n_1+1)-n_1} \times \mathbb{F}_2^{n_1}$:

$$\forall\ i\ R_i(\mathbf{b}, \mathbf{c}) = 0\,\mathbf{?}$$

Dinur's second

Let $p_1, \ldots, p_m$ be the input polys. Here we doesn't accurately compute G.

parameters: n, n_1

Repeat few times do the following:

- Generate $R_1, \ldots, R_{n_1+1}$ random lin. comb. of $p_1, \ldots, p_m$
- compute the truth table of $\tilde{G}$

1. $\forall\ (\mathbf{b}, \mathbf{c}) \in \mathcal{W}^{n-n_1}_{d(n_1+1)-n_1} \times \mathbb{F}_2^{n_1}$:

   $\forall\ i\ R_i(\mathbf{b}, \mathbf{c}) = 0$ **?**

   Yes: $\tilde{G}(\mathbf{b}) = 1$ **No**: $\tilde{G}(\mathbf{b}) = 0$

Dinur's second

Let $p_1, \ldots, p_m$ be the input polys. Here we doesn't accurately compute G.
parameters: n, n_1

Repeat few times do the following:

- Generate $R_1, \ldots, R_{n_1+1}$ random lin. comb. of $p_1, \ldots, p_m$
- compute the truth table of $\tilde{G}$

1. $\forall\ (\mathbf{b}, \mathbf{c}) \in \mathcal{W}^{n-n_1}_{d(n_1+1)-n_1} \times \mathbb{F}_2^{n_1}$:

   $\forall\ i\ R_i(\mathbf{b}, \mathbf{c}) = 0$ **?**

   Yes: $\tilde{G}(\mathbf{b}) = 1$ **No**: $\tilde{G}(\mathbf{b}) = 0$
2. Interpolate $\tilde{G}$.

Dinur's second

Let $p_1, \ldots, p_m$ be the input polys. Here we doesn't accurately compute G.

parameters: n, n_1

Repeat few times do the following:

- Generate $R_1, \ldots, R_{n_1+1}$ random lin. comb. of $p_1, \ldots, p_m$
- compute the truth table of $\tilde{G}$

1. $\forall\ (\mathbf{b}, \mathbf{c}) \in \mathcal{W}^{n-n_1}_{d(n_1+1)-n_1} \times \mathbb{F}_2^{n_1}$:

   $\forall\ i\ R_i(\mathbf{b}, \mathbf{c}) = 0$ **?**

   Yes: $\tilde{G}(\mathbf{b}) = 1$ **No**: $\tilde{G}(\mathbf{b}) = 0$

2. Interpolate $\tilde{G}$.

- For each $\mathbf{b} \in \mathbb{F}_2^{n-n_1}$ s.t $\tilde{G}(\mathbf{b}) = 1$ **somehow** compute $\mathbf{c} \in \mathbb{F}_2^{n-n_1}$ s.t

  $\forall\ i\ R_i(\mathbf{b}, \mathbf{c}) = 0$

Dinur's second

Let $p_1, \ldots, p_m$ be the input polys. Here we doesn't accurately compute G.

parameters: n, n_1

Repeat few times do the following:

- Generate $R_1, \ldots, R_{n_1+1}$ random lin. comb. of $p_1, \ldots, p_m$
- compute the truth table of $\tilde{G}$

  1. $\forall\ (\mathbf{b}, \mathbf{c}) \in \mathcal{W}^{n-n_1}_{d(n_1+1)-n_1} \times \mathbb{F}_2^{n_1}$:

     $$\forall\ i\ R_i(\mathbf{b}, \mathbf{c}) = 0\,\textbf{?}$$

     Yes: $\tilde{G}(\mathbf{b}) = 1$ **No**: $\tilde{G}(\mathbf{b}) = 0$

  2. Interpolate $\tilde{G}$.

- For each $\mathbf{b} \in \mathbb{F}_2^{n-n_1}$ s.t $\tilde{G}(\mathbf{b}) = 1$ **somehow** compute $\mathbf{c} \in \mathbb{F}_2^{n-n_1}$ s.t

  $$\forall\ i\ R_i(\mathbf{b}, \mathbf{c}) = 0$$

- if $(\mathbf{b}, \mathbf{c})$ showed up before, then check if $p_1(\mathbf{b}, \mathbf{c}) = \cdots = p_{n_1+1}(\mathbf{b}, \mathbf{c}) = 0$
- continue until one solution is found

Dinur's second

Let $p_1, \ldots, p_m$ be the input polys. Here we doesn't accurately compute G.
parameters: n, n_1

Repeat few times do the following:

- Generate $R_1, \ldots, R_{n_1+1}$ random lin. comb. of $p_1, \ldots, p_m$
- compute the truth table of $\tilde{G}$
  1. $\forall\ (\mathbf{b}, \mathbf{c}) \in \mathcal{W}^{n-n_1}_{d(n_1+1)-n_1} \times \mathbb{F}_2^{n_1}$:

     $\forall\ i\ R_i(\mathbf{b}, \mathbf{c}) = 0$ **?**

     Yes: $\tilde{G}(\mathbf{b}) = 1$ **No**: $\tilde{G}(\mathbf{b}) = 0$
  2. Interpolate $\tilde{G}$.
- For each $\mathbf{b} \in \mathbb{F}_2^{n-n_1}$ s.t $\tilde{G}(\mathbf{b}) = 1$ **somehow** compute $\mathbf{c} \in \mathbb{F}_2^{n-n_1}$ s.t

  $\forall\ i\ R_i(\mathbf{b}, \mathbf{c}) = 0$

- if $(\mathbf{b}, \mathbf{c})$ showed up before, then check if $p_1(\mathbf{b}, \mathbf{c}) = \cdots = p_{n_1+1}(\mathbf{b}, \mathbf{c}) = 0$
- continue until one solution is found

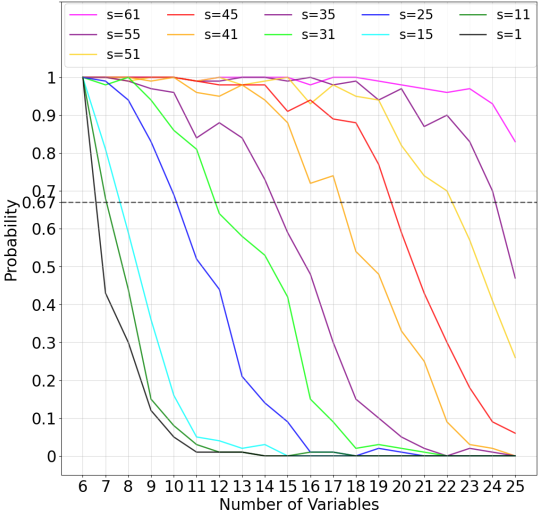
Complexity

complexity $= \begin{cases} O\left(n^2 \cdot 2^{0.815n}\right) \text{ if } d = 2, \textit{and} \\ O\left(n^2 \cdot 2^{(1-1/(2.7d))n}\right) \text{ if } d > 2 \end{cases}$

Practical results

Probability of success

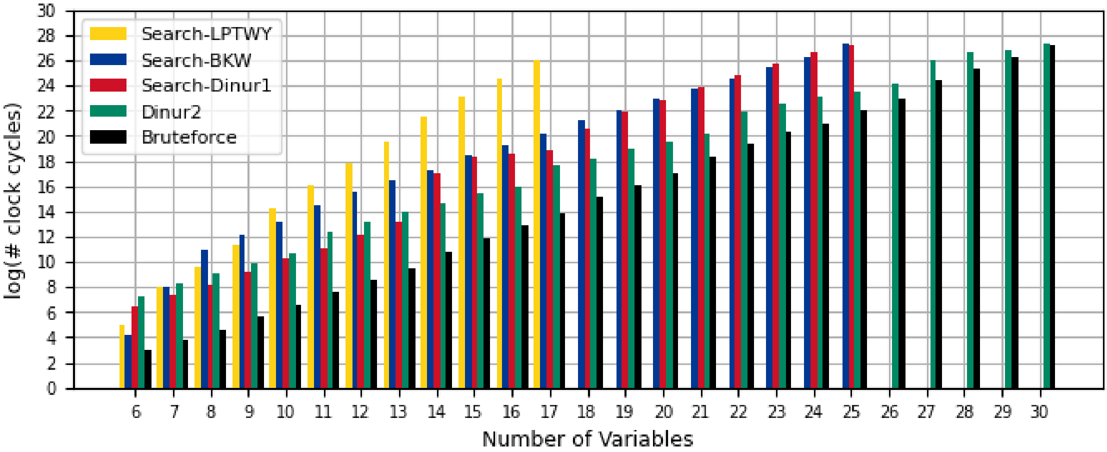
Björklund et al.'s with $\lambda = 0.1967$ and several values of s.



- $s = 48n + 1$ in Björklund and Dinur's first
- Internal iterations can be reduced!
- similar result for Dinur1
- A bit fluctuant for Lokshtanov (still small)
- Dinur2 always success probability $\geq$ 0.9

Practical times complexities

Cryptography
Research
Centre



Rate of growth and outperformance of bruteforce (BF)

Cryptography Research Centre

Table 1: Growth rate of the practical complexity of **solving a square quadratic** system with at most one solution. In the first three rows, it means with probability of success greater than $2/3$.

Algorithm	n_{max}	Experimental $(14 \leq n \leq n_{max})$	Theoretical $(n \to \infty)$	Beat (BF) for $n \geq$
Lokshtanov et al.'s	17	$2^{0.912}$	$2^{0.876}$	129
Björklund et al.'s	25	$2^{0.876}$	$2^{0.804}$	60
Dinur's first	25	$2^{0.971}$	$2^{0.694}$	132
Dinur's second	30	$2^{0.818}$	$2^{0.815}$	33
Bruteforce	30	$2^{1.022}$	2^{1}	

Future work?

Future work?

1. Fast implementations?

Future work?

1. Fast implementations?
2. Parallel implementations (on GPUs)? How deal with memory access cost?

Future work?

Cryptography Research Centre

1. Fast implementations?
2. Parallel implementations (on GPUs)? How deal with memory access cost?
3. Quantum versions of the algorithms?

Technology Innovation Institute

Cryptography Research Centre

References I

[Beu20] Ward Beullens. Sigma protocols for MQ, PKP and SIS, and fishy signature schemes. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 183–211, Cham, 2020. Springer International Publishing.

[Beu21] Ward Beullens. Mayo: Practical post-quantum signatures from oil-and-vinegar maps. In Andreas Hülsing and Riham AlTawy, editors, *Selected Areas in Cryptography*. Springer International Publishing, 2021.

[CFMR+17] A. Casanova, J.-C. Faugère, G. Macario-Rat, J. Patarin, L. Perret, and J. Ryckeghem. GeMSS: A great multivariate short signature. NIST CSRC, 2017. Official website: `https://www-polsys.lip6.fr/Links/NIST/GeMSS.html`.

References II

[CHR+18] Ming-Shing Chen, Andreas Hülsing, Joost Rijneveld, Simona Samardjiska, and Peter Schwabe. SOFIA: $\mathcal{MQ}$-based signatures in the qrom. In Michel Abdalla and Ricardo Dahab, editors, *Public-Key Cryptography – PKC 2018*, pages 3–33, Cham, 2018. Springer International Publishing.

[CHR+20] Ming-Shing Chen, Andreas Hülsing, Joost Rijneveld, Simona Samardjiska, and Peter Schwabe. MQDSS specifications, 2020. `http://mqdss.org/specification.html`.

[DCP+17] J. Ding, M.S. Chen, A. Petzoldt, D. Schmidt, and B.Y. Yang. Rainbow. NIST CSRC, 2017. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/submissions/Rainbow.zip.

[Deu85] David Deutsch. Quantum Theory, the Church–Turing Principle and the Universal Quantum Computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, 1985.

Valiant–Vazirani affine hashing

- it isolates one solution to the system
- add k random linear equations to original system
- $k = \log |S|$, where S is the set of solutions.
- For the probability $\Pr[U_x]$ that $x \in \mathcal{S}$ is the only solution, we have

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

Therefore

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

- repeat this up to $8n \log n$ times. With probability $1 - 1/n$ some of the solutions would be isolated.

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