arXiv:2603.05054v1 [cs.CC] 5 Mar 2026

攻击有限域迷宫中的多项式问题*

´Angela Barbero[1],Ragnar Freij-Hollanti[2],Camilla Hollanti[2],H˚avard Raddum[3],Øyvind Ytrehus[4],和 Morten Øygarden[3,4]

[1]巴利亚多利德大学，西班牙[2]阿尔托大学，芬兰[3]Simula UiB，挪威[4]卑尔根大学，挪威

摘要

2025年4月，GMV宣布了一项竞赛，旨在寻找求解有限域上某一特定多项式系统的最佳方法。本文提出一种方法，其求解给定方程组的速度显著快于暴力搜索或标准Gr¨obner基方法所能达到的速度。该方法利用多项式系统的结构化稀疏性，通过相继计算 $resultants$，求出关联理想中的一个单变量多项式。随后可从该单变量多项式高效地恢复该系统的一个解。文中给出了所提出的ResultantSolver算法的伪代码，以及实验和对竞争方法的比较。我们还讨论了进一步的潜在改进，例如将部分计算并行化。

1 背景

2025年4月，GMV 公司举办了一场竞赛，旨在寻找求解具有特定结构的多项式方程组的有效方法。该方程组定义在有限域 $\mathbb{F}_p$ 上，方程组中的变量数量与 p 的位数相关。GMV 非常了解求解有限域上非线性方程组的标准方法，但希望找到能够利用该方程组特殊结构的技术。超过 100 名参赛者参加了竞赛，获奖者于 2025年7月公布

*本研究部分得到了芬兰研究委员会资助（编号：#351271）、芬兰商务促进局商业芬兰联合创新联盟资助（编号：#BFRK/473/31/2024, 5845483），以及由 MICIU/AEI/10.13039/501100011033 和 FEDER（欧盟）资助的 R&D+i 项目 PID2021-124613OB-I00 的支持。

[4]。本文介绍我们为求解 GMV 系统而开发的方法。

我们求解 [5] 中给出的多项式方程组的策略，是利用该系统的特定结构，因为这种结构适用于结式理论。这一思路受到近期密码分析工作的启发：在这些工作中，研究人员利用结式求解大有限域上某些置换的受限输入/受限输出（CICO）问题，其效率高于使用通用求解方法 [7, 1]。具体而言，在这种情况下，CICO 问题归结为求解大素数域上的一个结构化多元多项式系统。

结式以两个多项式 f 和 g 以及一个目标变量 x_i 作为输入，并生成一个消去了该目标变量的新多项式:

$$\mathrm{Res}(f(x_1,\ldots,x_n), g(x_1,\ldots,x_n); x_i) = h(x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n).$$

一个有用的性质是，新多项式会保留 f 和 g 的公共根。结式通过计算关于 f 和 g 的西尔维斯特矩阵的行列式得到，构造过程如下。设 $\deg(f) = d$，且 $\deg(g) = e$，并将 f 和 g 写为

$$\begin{aligned} f &= \alpha_d x_i^d + \alpha_{d-1} x_i^{d-1} + \ldots + \alpha_1 x_i + \alpha_0 \\ g &= \beta_e x_i^e + \beta_{e-1} x_i^{e-1} + \ldots + \beta_1 x_i + \beta_0, \end{aligned}$$

其中 α_j 和 β_j 属于 $\mathbb{F}_p[x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n]$。$f$ 和 g 的结式计算为 $(d+e) \times (d+e)$ 行列式

$$\mathrm{Res}(f,g;x_i) = \begin{vmatrix} \alpha_d & \alpha_{d-1} & \cdots & \alpha_0 & 0 & \cdots & 0 \\ 0 & \alpha_d & \alpha_{d-1} & \cdots & \alpha_0 & \cdots & 0 \\ \ddots & \ddots & & & \ddots & \ddots & \\ 0 & \cdots & 0 & \alpha_d & \alpha_{d-1} & \cdots & \alpha_0 \\ \beta_e & \beta_{e-1} & \cdots & \beta_0 & 0 & \cdots & 0 \\ 0 & \beta_e & \beta_{e-1} & \cdots & \beta_0 & \cdots & 0 \\ \ddots & \ddots & & & \ddots & \ddots & \\ 0 & \cdots & 0 & \beta_e & \beta_{e-1} & \cdots & \beta_0 \end{vmatrix}. \tag{1}$$

2 GMVPolynomialSystem

GMV 挑战中给出的系统定义在 $\mathbb{F}_p[x_0,\ldots,x_n]$ 上，如下所示:

$$\begin{aligned} f_0 = \;& b_1 x_0 + x_1 + x_n = 0 \\ f_1' = \;& a_0 x_0^3 - (b_0 x_0 + 2x_1)(a_1 x_0^2 + x_1^2 + 1) = 0 \\ f_i = \;& x_i(x_{i-1}^2 - 1) + (a_i x_1 + b_i x_n)(2x_i x_{i-1} - x_{i-1}^2 + 1) + 2x_{i-1} = 0, \\ & \text{for } 2 \le i \le n-1 \\ f_n = \;& t(x_{n-1}^2 + 2x_n x_{n-1} - 1) - x_n(x_{n-1}^2 - 1) + 2x_{n-1} = 0, \end{aligned} \tag{2}$$

其中常数为 t，a_j，$b_j \in \mathbb{F}_p$。文中 [5]并未说明这些常数的具体选取方式。因此，在本文余下部分，我们将假设这些常数是一般的；在自行生成的实验中，则从 $\mathbb{F}_p$ 中均匀随机抽取这些常数。

注意到，上述多项式系统特别适合使用结式。事实上，对于 $i = 2, \ldots, n-1$，变量 x_i只出现在两个多项式 f_i 和 f_{i+1}中，因此可以通过一次结式计算将其从系统中消去。更正式地说，对于理想$I = \langle f_0, f_1', f_2, \ldots f_n\rangle$，我们有

$$I \cap \mathbb{F}_p[x_1, \ldots, x_{i-1}, x_{i+1}, \ldots x_n] = \langle f_0, f_1', f_2, \ldots, f_{i-1}, g_i, f_{i+2} \ldots f_n\rangle,$$

其中 $g_i = Res(f_i, f_{i+1}; x_i)$。现在可以利用 x_{i-1} 以及多项式 f_{i-1} 和 g_i继续这一过程，依此类推。本文余下部分使用的一种记号：设变量 x_i 在任意多项式 $f \in \mathbb{F}[x_1, \ldots, x_n]$中的次数记为 $\deg_{x_i}(f)$。

2.1 初始准备

我们对给定系统作如下初始化和略微修改。首先，使用$b_1x_0 + x_1 + x_n = 0$从 f_1' 中消去 x_0，并将所得多项式记为f_1:

$$f_1(x_1, x_n) = f_1'(-b_1^{-1}(x_1 + x_n), x_1).$$

该多项式 f_1 随后包含

$$(-a_0/b_1^3 + a_1b_0/b_1^3 - 2a_1/b_1^2 + b_0/b_1 - 2)x_1^3$$

作为 x_1 中的三次项，而 f_1 中的所有其他项在 x_1中的次数均为 ≤ 2 。原则上，上述系数可能计算为零，但对于来自大型域的随机系数而言，这种情况不太可能发生。

从 $f_1 = 0$ 中，我们将纯 x_1^3项单独取出，记为 $x_1^3 = r(x_1, x_n)$，其中 $\deg_{x_1}(r) \leq 2$。随后，我们可以在环 $\mathbb{F}_p[x_1, \ldots, x_n]/\langle x_1^3 - r(x_1, x_n)\rangle$ 中执行所有不涉及 f_1 的计算，从而确保任何计算输出中的 x_1 次数始终小于 3。

2.2 高层算法

求解该系统的高层算法是通过结式计算逐一消去变量x_{n-1}、x_{n-2}、……、x_2。最后剩下 $f_1(x_1, x_n)$和 $g_2(x_1, x_n)$，其中 $g_2(x_1, x_n)$是经过 $n-2$ 次结式计算后得到的最终结果。然后进行最后一次结式计算 $\mathrm{Res}(f_1(x_1, x_n), g_2(x_1, x_n); x_1) = u(x_n)$，得到一个单变量多项式 $u(x_n)$，它属于理想 $\langle f_1, f_2, \ldots\ldots, f_n\rangle$。现在可以用预期 $O(n\log^2 n\log pn)$次乘法的时间复杂度求出 $u(x_n)$的根，见 $\mathbb{F}_p$ [6, 推论 14.16 和第 8]章。正如 [5], 中所述，一旦知道 x_n 的正确取值，恢复完整解就很简单。

2.3 示例案例

我们通过挑战中给出的例子来说明基本求解算法的工作原理，其中 $p = 1523$ 和 $n = 7$。为更清楚地展示这一过程及计算过程中发生的情况，我们不逐一写出多项式f_1，……，f_7，而是用它们在每个单独变量上的次数向量 δ_i 来表示。也就是说，δ_i 表示为

$$\delta_i = (\deg_{x_1}(f_i), \deg_{x_2}(f_i), \ldots, \deg_{x_7}(f_i)).$$

初始化之后、尚未计算任何结式时的系统如下所示

polynomial	$\delta_i = (\deg_{x_1}, \ldots, \deg_{x_7})$
f_1	$(3,0,0,0,0,0,3)$
f_2	$(3,1,0,0,0,0,1)$
f_3	$(1,2,1,0,0,0,1)$
f_4	$(1,0,2,1,0,0,1)$
f_5	$(1,0,0,2,1,0,1)$
f_6	$(1,0,0,0,2,1,1)$
f_7	$(0,0,0,0,0,2,1)$

我们首先取 f_6 与 f_7的结式，消去 x_6:

$$g_6(x_1, x_5, x_7) = \mathrm{Res}(f_6, f_7; x_6).$$

计算该结式是通过计算一个 3×3 行列式来完成的。将 f_7写成 $f_7 = \alpha_2 x_6^2 + \alpha_1 x_6 + \alpha_0$，其中每个 $\alpha_i \in \mathbb{F}_p[x_7]$，并将 f_6 写成 $f_6 = \beta_1 x_6 + \beta_0$，其中 $\beta_i \in \mathbb{F}_p[x_1, x_5, x_7]$。然后

$$g_6(x_1, x_5, x_7) = \begin{vmatrix} \beta_1 & \beta_0 & 0 \\ 0 & \beta_1 & \beta_0 \\ \alpha_2 & \alpha_1 & \alpha_0 \end{vmatrix} = \alpha_2\beta_0^2 - \alpha_1\beta_1\beta_0 + \alpha_0\beta_1^2.$$

此后，系统如下所示

pol	$(\deg_{x_1}, \ldots, \deg_{x_7})$
f_1	$(3,0,0,0,0,0,3)$
f_2	$(3,1,0,0,0,0,1)$
f_3	$(1,2,1,0,0,0,1)$
f_4	$(1,0,2,1,0,0,1)$
f_5	$(1,0,0,2,1,0,1)$
g_6	$(2,0,0,0,4,0,3)$

其中，我们将所有多项式乘法都对 $x_1^3 - r(x_1, \ldots, x_n)$ 取模，以保持 $\deg_{x_1}(g_6) \leq 2$。我们看到，$\deg_{x_5}(g_6) = 4$，是 $\deg_{x_5}(f_6)$ 的两倍。这是该系统的一般特性，我们将在下文证明；这也是求解算法的整体复杂度达到 $\mathcal{O}(2^n)$ 的原因。同样，我们还有 $\deg_{x_7}(g_6) = \deg_{x_5}(g_6) - 1$。我们也将证明这一关系普遍成立。

接下来，我们通过递归计算 $g_5 = \text{Res}(f_5, g_6; x_5)$，直到 $g_2 = \text{Res}(f_2, g_3; x_2)$。这些行列式不再是 3×3 行列式，但它们是稀疏的，因为在消去 x_i 时，$\deg_{x_i}(f_i) = 1$。对于每个结式，我们有[1] $f_i = \beta_1 x_i + \beta_0$ 和 $g_{i+1} = \alpha_{2^{7-i}} x_i^{2^{7-i}} + \alpha_{2^{7-i}-1} x_i^{2^{7-i}-1} + \ldots + \alpha_1 x_i + \alpha_0$，其中 α_j, $\beta_j \in \mathbb{F}_p[x_1, x_{i-1}, x_7]$。接下来要计算的 $(2^{7-i}+1) \times (2^{7-i}+1)$ 行列式为

$$\text{Res}(f_i, g_{i+1}; x_i) = \begin{vmatrix} \beta_1 & \beta_0 & 0 & 0 & \cdots & 0 \\ 0 & \beta_1 & \beta_0 & 0 & \cdots & 0 \\ \vdots & \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 0 & \beta_1 & \beta_0 \\ \alpha_{2^{7-i}} & \alpha_{2^{7-i}-1} & \alpha_{2^{7-i}-2} & \cdots & \alpha_1 & \alpha_0 \end{vmatrix}. \quad (3)$$

我们将在后文证明，由于这些结式具有稀疏性，因此可以高效地计算它们。依次计算 $g_i = \text{Res}(f_i, g_{i+1}; x_i)$ for $i = 5$，4，3，2 后，我们得到如下系统，其中持续使用了 $x_1^3 = r$，以保持 $\deg_{x_1} \leq 2$。

pol	$(\deg_{x_1}, \ldots, \deg_{x_7})$
f_1	$(3, 0, 0, 0, 0, 0, 3)$
g_2	$(2, 0, 0, 0, 0, 0, 127)$

degree $\deg_{x_7}(g_2)$ 变为 127 而不是 63，这是因为 f_2 是一个特殊情形，其 $\deg_{x_1}(f_2) = 3$，其中 $i > 2$ 的 f_i 的 $\deg_{x_1}(f_i) = 2$。因此，使用 $r(x_1, x_7)$ 进行约化会再次将 x_7 中的次数加倍，以保持 $\deg_{x_1}(g_2) \leq 2$。现在，我们还需要在 f_1 与 g_2 之间进行最后一次结式运算，以消去 x_1 并求出 $u(x_7)$。该结式不属于式 (3) 中的稀疏类型，但由于 $\deg_{x_1}(f_1) = 3$ 且 $\deg_{x_1}(g_2) = 2$，它只会是一个 5×5 行列式。计算这个最终结式后，我们得到 $u(x_7)$ 在此情形下的唯一根 $x_7 = 180$。

需要注意的是，我们计算结式的顺序对复杂度影响很大。观察由此得到的树结构最能说明这一点，如图 1 和图 2 所示。

3 结式求解方法在一般系统中的行为

本节将展示并证明结式的一些有用性质，这些性质适用于式 (2) 中的特定系统。本节最后将证明，理想 $\langle f_1$、……、$f_n \rangle$ 中的单变量多项式 $u(x_n)$ 的次数为 $3(2^n - 1)$。

[1] 多项式 β_0、β_1 和 α_j 取决于 i，但为了使记号更简洁，在没有混淆风险时，我们省略 i。

命题 1。与式 *(3)* 中类似的一般行列式为

$$\det(\alpha,\beta;h) := \begin{vmatrix} \beta_1 & \beta_0 & 0 & \cdots & 0 \\ \ddots & \ddots & & & \ddots \\ 0 & 0 & \cdots & \beta_1 & \beta_0 \\ \alpha_h & \alpha_{h-1} & \cdots & \alpha_1 & \alpha_0 \end{vmatrix} = \sum_{i=0}^{h} (-1)^i \alpha_i \beta_0^i \beta_1^{h-i}. \quad (4)$$

证明：我们通过对 h 使用归纳法来证明该命题。对于 $h = 1$，该公式显然成立，因为

$$\begin{vmatrix} \beta_1 & \beta_0 \\ \alpha_1 & \alpha_0 \end{vmatrix} = \alpha_0\beta_1 - \alpha_1\beta_0 = \sum_{i=0}^{1} (-1)^i \alpha_i \beta_0^i \beta_1^{1-i}.$$

假设 $\det(\alpha,\beta;\ h-1) = \sum_{i=0}^{h-1} (-1)^i \alpha_i \beta_0^i \beta_1^{h-1-i}$（归纳假设）。则 $\det(\alpha,\beta;\ h)$ 可通过沿最左列进行余子式展开来计算，如下所示

$$\begin{aligned} \det(\alpha,\beta;h) &= \beta_1 \cdot \begin{vmatrix} \beta_1 & \beta_0 & 0 & \cdots & 0 \\ \ddots & \ddots & & & \ddots \\ 0 & 0 & \cdots & \beta_1 & \beta_0 \\ \alpha_{h-1} & \alpha_{h-2} & \cdots & \alpha_1 & \alpha_0 \end{vmatrix} \\ &+ (-1)^h \alpha_h \cdot \begin{vmatrix} \beta_0 & 0 & 0 & \cdots & 0 \\ \beta_1 & \beta_0 & 0 & \cdots & 0 \\ \ddots & \ddots & & & \ddots \\ 0 & 0 & \cdots & \beta_1 & \beta_0 \end{vmatrix} \end{aligned}$$

第一个行列式为 $\det(\alpha,\beta;\ h-1)$，而第二个行列式等于 β_0^h，因为它是对角线上为 β_0 的对角（$h \times h$）矩阵。因此，利用归纳假设，该表达式可化为

$$\begin{aligned} \beta_1 \sum_{i=0}^{h-1} (-1)^i \alpha_i \beta_0^i \beta_1^{h-1-i} + (-1)^h \alpha_h \beta_0^h &= \sum_{i=0}^{h-1} (-1)^i \alpha_i \beta_0^i \beta_1^{h-i} + (-1)^h \alpha_h \beta_0^h \\ &= \sum_{i=0}^{h} (-1)^i \alpha_i \beta_0^i \beta_1^{h-i} \end{aligned}$$

即为所求。 □

推论 1. 结式 $g_i = \mathrm{Res}(f_i, g_{i+1}; x_i)$ 可用 $4h$ 次多项式乘法计算，其中 h 是多项式 g_{i+1} 中 x_i 的次数

证明：多项式序列 $\beta_0^2, \ldots, \beta_0^h$ 和 $\beta_1^2, \ldots, \beta_1^h$ 各需 $h-1$ 次乘法即可计算得出。命题 1 中求和的每一项只需 2 次乘法即可计算，因此整个求和只需 $2(h+1)$ 次乘法即可计算。总之，计算 $\mathrm{Res}(f_i, g_{i+1}; x_i)$ 需要 $2(h-1) + 2(h+1) = 4h$ 次乘法。 □

命题 2。 设 $a \geq 3$，并假设变量 $x_{n-1}, \ldots, x_a$ 已通过结式消去，从而得到多项式 $g_n = f_n, g_{n-1}, \ldots, g_{a+1}$，其中 $g_i = \mathrm{Res}(f_i, g_{i+1}; x_i)$，对于 $3 \leq i \leq a$。那么$\deg_{x_{a-1}}(g_a) = 2^{n-a+1}$ 且$\deg_{x_n}(g_a) = 2^{n-a+1} - 1$。

证明： 我们对 a 进行归纳法来证明该命题。归纳基础：对于 $a = n$（也就是说，还没有消去任何变量），该命题成立，因为系统的构造保证了 $\deg_{x_{n-1}}(f_n) = 2 = 2^{n-n+1}$ 且 $\deg_{x_n}(f_n) = 1 = 2^{n-n+1} - 1$。

归纳步骤：假设变量 $x_{n-1}, \ldots, x_{a+1}$ 已被消去，且 $\deg_{x_a}(g_{a+1}) = 2^{n-a}$，同时 $\deg_{x_n}(g_{a+1}) = 2^{n-a} - 1$。由于 $\deg_{x_a}(g_{a+1}) = 2^{n-a}$，可以按照命题 1 中的方法，使用 $h = 2^{n-a}$ 进行结式计算。此外，$\deg_{x_{a-1}}(g_{a+1}) = 0$，因为变量 x_{a-1} 未参与任何消去 $x_{n-1}, \ldots, x_{a+1}$ 的结式。根据初始系统的构造可知，将 f_a 写成 $f_a = \beta_1 x_a + \beta_0$ 时，对于 $i = 1, 2$，有 $\deg_{x_{a-1}}(\beta_i) = 2$ 且 $\deg_{x_n}(\beta_i) = 1$。我们计算命题 1 中求和式内 x_{a-1} 和 x_n 各项的次数：

$$\deg_{x_{a-1}}(\alpha_i \beta_0^i \beta_1^{2^{n-a}-i}) = 0 + 2i + 2(2^{n-a} - i) = 2^{n-a+1}$$
$$\deg_{x_n}(\alpha_i \beta_0^i \beta_1^{2^{n-a}-i}) = (2^{n-a} - 1) + i + (2^{n-a} - i) = 2^{n-a+1} - 1.$$

由此通过归纳法证明了这一结论。 □

我们可以依次消去一个变量，从 x_{n-1} 开始，到 x_3结束，方法是计算 $g_i = \mathrm{Res}(f_i, g_{i+1}; x_i)$，其中 $i = n-1, \ldots, 3$。消去 x_3后，我们得到一个由三个多项式 f_1, f_2, g_3 组成的系统，其变量为 x_1, x_2, x_n。根据命题 2，该系统在各个变量上的次数如下：

pol	$(\deg_{x_1}, \ldots, \deg_{x_n})$
f_1	$(3, 0, 0, \ldots, 0, 3)$
f_2	$(3, 1, 0, \ldots, 0, 1)$
g_3	$(2, 2^{n-2}, 0, \ldots, 0, 2^{n-2} - 1)$.

(5)

由于 $\deg_{x_1}(f_2) = 3$，因此无法根据命题 2 计算下一个结式的次数。

注 *1*。我们将 $deg_{x_1, x_n}(f)$ 定义为 f 关于变量 x_1 和x_n的联合次数，也就是说，将所有变量 x_i、 $i \neq 1$、 n替换为常数后， f 的总次数。例如，$deg_{x_1, x_n}(x_1^k x_n^k) = 2k$，但 $deg_{x_1, x_n}((x_1 + x_n)^k) = k$，尽管这两种情况下， x_1 和 x_n 的次数均为 k。仿照命题 *2* 中的方法，通过归纳法并仔细计算每个 α_i 项和 β_i 项的次数，特别考虑到对于所有 $a = n - 1, \ldots, 3$，所有 f_a 中的 β项均有 $deg_{x_1, x_n} \beta_i = 1$，我们可以推得 $deg_{x_1, x_n}(g_a) = 2^{n-a+1} - 1$。

命题 3. 设 $g_2 = \mathrm{Res}(f_2, g_3; x_2)$ 其中次数轮廓如 *(5)* 中所示。则 $\deg_{x_n}(g_2) = 2^n - 1$。

证明：如前所述，这一情形与之前的情形略有不同，因为 $\deg_{x_1}(f_2)=3$，但我们将像之前一样进行拆分，

$$f_2=\beta_1x_2+\beta_0,\ \text{ and }\ g_3=\sum_{i=0}^{2^{n-2}}\alpha_ix_2^i$$

其中 $\alpha_i,\beta_j\in\mathbb{F}_p[x_1,x_n]$。仔细研究可知，$\deg_{x_1}(\beta_1)=2$ and $\deg_{x_n}(\beta_1)=1$，而 $\deg_{x_1,x_n}(\beta_1)=2$。此外，$\deg_{x_1}(\beta_0)=3$and $\deg_{x_n}(\beta_0)=1$，而 deg $_{x_1\ x_n}\ \beta_0=3$ 1, $_{x_1\ x_n}\ \alpha_i\ \leq$,( )。最后，利用备注中说 2^{n-2} $i=0\ldots\ 2^{n-2}$ 明的推理，对所有 ,, 有 deg,( )。

因此

$$\deg_{x_1,x_n}(\alpha_i\beta_0^i\beta_1^{2^{n-2}-i})\leq 2^{n-2}-1+3i+2(2^{n-2}-i)=2^{n-2}+2^{n-1}+i-1.$$

由于 $i\leq 2^{n-2}$，我们有

$$\deg_{x_1,x_n}(\alpha_i\beta_0^i\beta_1^{2^{n-2}-i})\leq 2^{n-2}+2^{n-1}+2^{n-2}-1=2^n-1.$$

这表明 g_2 关于变量 x_1 和 x_n 的总次数至多为 2^n-1。如果接着应用约化多项式 $r(x_1,x_n)$ 来降低 x_1 中的次数，那么 x_1 和 x_n 中的总次数不会改变，因为 $\deg_{x_1,x_n}(r(x_1,x_n))=3=\deg_{x_1,x_n}(x_1^3)$。

经过化简后，我们仍有 $\deg_{x_1,\ x_n}(g_2)\leq 2^n-1$，这意味着 $\deg_{x_n}(g_2)\leq 2^n-1$，证明完毕。

□

求得关于$\langle f_1$、.....、$f_n\rangle$的单变量多项式之前，需要计算的最后一个结式是 $\mathrm{Res}(f_1,\ g_2;\ x_1)$。这个结式并非稀疏型，因为 $\deg_{x_1}(f_1)=3$ 且 $\deg_{x_1}(g_2)=2$，而命题 1 仅适用于其中一个多项式关于待消去变量为一次的情形。不过，由于 f_1 和 g_2 的 x_1次数分别只有 2 和 3，这个最后的结式只需计算一个 5×5 矩阵的行列式。令 $f_1=\beta_3x_1^3+\beta_2x_1^2+\beta_1x_1+\beta_0$ 和$g_2=\alpha_2x_1^2+\alpha_1x_1+\alpha_0$。那么该行列式的形式如下：

$$u(x_n)=\mathrm{Res}(f_1,g_2;x_1)=\begin{vmatrix}\beta_3&\beta_2&\beta_1&\beta_0&0\\0&\beta_3&\beta_2&\beta_1&\beta_0\\\alpha_2&\alpha_1&\alpha_0&0&0\\0&\alpha_2&\alpha_1&\alpha_0&0\\0&0&\alpha_2&\alpha_1&\alpha_0\end{vmatrix}\tag{6}$$

通过检查 f_1 的分解，我们发现 $\deg_{x_n}(\beta_i)=3-i$。同样，我们已在命题 3 中证明，g_2 中最高次数单项式的总次数为 2^n-1。也就是说，$\deg(\alpha_2x_1^2)=\deg(\alpha_1x_1)=\deg(\alpha_0)=2^n-1$。这意味着 $\deg_{x_n}(\alpha_j)=2^n-1-j$。利用这些次数，我们便能证明以下命题。

命题 4。 u（x_n）在（6）中的次数为 $3\cdot(2^n-1)$。

证明： （6）中的行列式是通过将所有元素的乘积求和来计算的，其中每一行和每一列恰好选取一个元素。也就是说，每个不涉及矩阵中任何 0 的乘积都具有以下形式： $\beta_{i_1}\beta_{i_2}\alpha_{j_1}\alpha_{j_2}\alpha_{j_3}$。通过观察或繁琐的计算，我们可以看到，对所有可能的非零乘积都成立的不变量是 $i_1+i_2+j_1+j_2+j_3=6$。利用已知的次数 $\deg_{x_n}(\beta_i) = 3-i$ 以及 $\deg_{x_n}(\alpha_j) = 2^n-1-j$，我们发现每个非零项的次数，从而 $u(x_n)$ 的次数为

$$(3-i_1)+(3-i_2)+(2^n-1-j_1)+(2^n-1-j_2)+(2^n-1-j_3) = \\ 3\cdot 2^n+3-(i_1+i_2+j_1+j_2+j_3) = 3\cdot 2^n+3-6 = 3\cdot(2^n-1)$$

□

在极不可能出现的情况下，如果计算中涉及的任何多项式的首项系数最终为 0，那么它唯一可能造成的影响就是降低最终次数，从而也降低整体复杂度。

无论是结式方法，还是通过计算 Gr¨obner 基来求解 (2)，都试图在由多项式 $f_1,\dots,f_n$ 张成的理想中找到单变量多项式 $u(x_n)$。u 的次数由命题 4 给出，这表明仅将其写入内存就需要数量级为 $\mathcal{O}(2^n)$ 的时间和空间复杂度。这为以下猜想提供了依据。

猜想 1. 任何通过在理想$\langle f_1,\dots,f_n\rangle$ 中寻找单变量多项式来求解方程组 *(2)* 的算法，其时间和空间复杂度至少为 $\mathcal{O}(2^n)$。

4 ResultantSolver算法与实现方面的考量

本节讨论用于求解主问题的算法的实现细节。第4.1–4.3节介绍并总结算法 1的实现细节，并简要讨论其时间复杂度与空间复杂度。最后，第4.4节将算法 1的性能与使用Magma的先进Gr¨obner基方法进行比较。

4.1 预计算

根据 [5], “.....任何能够为给定值提供解的方法 x_n ，并且 t基于其他值对应的某些已知解的数量 x_n ， t 也可能极其有用。”。尽管我们并不基于给定 t 值的特定解集来处理“其他 t值”的求解过程，但我们注意到， t 只出现在 f_n中。因此，由于变量是通过逐个应用方程迭代消去的，如果我们直到最后一个消元步骤才使用 f_n ，那么此前的所有消元步骤及其结果都对 t的任意值有效，并且可以视为联合预计算。算法 1 对此进行了描述

。如第 3 节的表 3 所示，执行时间可以分为预计算部分（对 t的所有值均有效）和 t特定的部分。事实上，对于我们所考察的情况，最后一步的执行时间相比预计算而言很短。

4.2 结式计算

直接地，变量 x_{n-1}，x_{n-2}、……、x_2可以按降序迭代消去。第 3 节中描述的算法正是这样做的：在计算 $g_i = \mathrm{Res}(f_i, g_{i+1}; x_i)$ 时，对 i 进行反向归纳。随后，可以用二叉树图形化地描述这一计算，如图 1 所示。在图 1 中，$g_i \in \mathbb{F}_p[x_1, x_{i-1}, x_n]$ 是 $f_i \in \mathbb{F}_p[x_1, x_{i-1}, x_i, x_n]$ 和 $g_{i+1} \in \mathbb{F}_p[x_1, x_i, x_n]$关于 x_i 的结式。

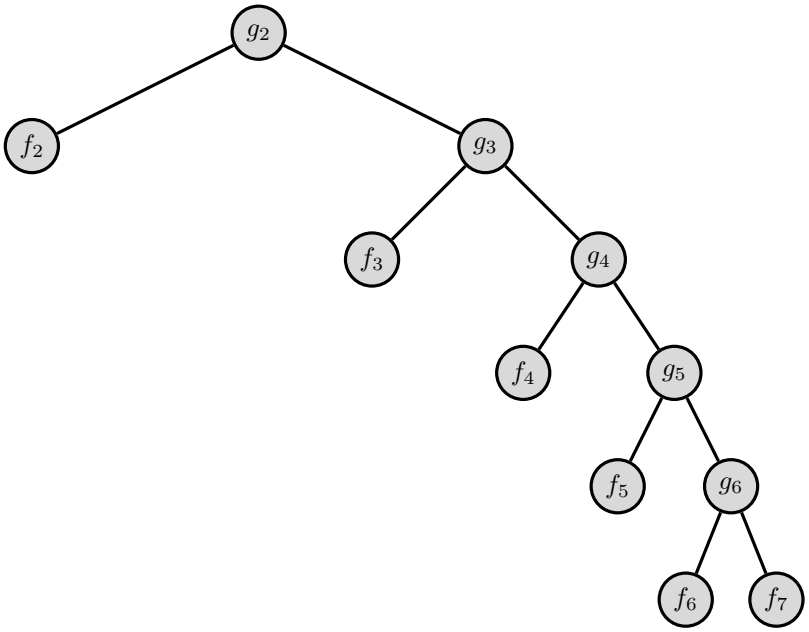

图 1：描述第 3 节算法的二叉树，其中包含 $n = 7$。

然而，计算结式的顺序是任意的，并且可以针对多项式列表 $f_1, \ldots f_n$ 上的任意二叉树结构执行该算法的推广版本。最终单变量多项式 $u(x_n)$ 是 $\langle f_1, \ldots, f_n \rangle \cap \mathbb{F}_p[x_n]$ 中唯一的最小多项式，这意味着结式的计算顺序不会影响代数，因此第 3 节中的命题可以适用于任意此类顺序。基于此，我们通过记作 $f_{[i,i]} = f_i$，将第 3 节中的构造推广为……

$1 \le i \le j \le k < n$，引入多项式

$$f_{[i+1,k]} = \mathrm{Res}(f_{[i+1,j]}, f_{[j+1,k]}; x_j) \in \mathbb{F}_p[x_1, x_i, x_k, x_n].$$

`特别地，按照第 3 节的术语，我们有 $g_i = f_{[i,\ldots n]}$。先验地，这一定义可能取决于所选的 j，但通常并非如此；尤其是在 $f_{[i+1,k]}$ 是 $\langle f_{i+1}, \ldots, f_k\rangle \cap \mathbb{F}_p[x_1, x_i, x_k, x_n]$ 中唯一的最小多项式这一一般情形下，更是如此。此外，无论情况是否如此，结式方程的任意解都可以通过回代得到原方程组的一个解。

如第 4.1 节所述，如果我们可能希望对 t的不同取值使用同一个多项式系统，那么我们希望将不依赖于 t的$f_{[2,n-1]}$的计算视为预计算。此外，为了在中间计算过程中降低全次数，从启发式角度来看，合理的做法是仅在计算的最后一步中引入多项式 f_1，因为它关于 x_n的次数为 3。剩余结式的组合顺序不会影响所需结式的数量。然而，涉及的行列式规模会随着所组合的原始多项式 f_i 的数量呈指数增长，因此，最好尽可能多地将计算安排在二叉树的叶节点附近。这意味着，除了直到最后一步才引入 f_1 和 f_n 之外，我们还希望计算所构成的二叉树尽可能平衡。

对性能的实测观察证实，在我们考察的这些情况下，采用平衡二叉树结构计算结式，可以减少内存使用并缩短执行时间，如图 2 和图 3 所示。为清晰起见，在图 2、图 3 以及第 4.3 节对算法 1 的讨论中，我们会同时写出多项式及其参数 x_U；x_U 是 $\{x_i\colon i \in U\}$ 的简写。每个节点中都对算法下一步要消去的变量的索引加了下划线。

在下面的例子中，我们给出与图 2 对应的第一组结式多项式。由于采用了平衡树结构，这些结式可以并行计算。

例1. 系统已初始化：$n = 7$, p，共有 11 比特。第一对多项式：

$$\begin{aligned} f_5 &= 1279x_1x_4^2 + 488x_1x_4x_5 + x_4^2x_5 + 1456x_4^2x_7 + 134x_4x_5x_7 \\ &\quad +244x_1 + 2x_4 + 1522x_5 + 67x_7 \\ f_6 &= 92x_1x_5^2 + 1339x_1x_5x_6 + x_5^2x_6 + 664x_5^2x_7 + 195x_5x_6x_7 \\ &\quad +1431x_1 + 2x_5 + 1522x_6 + 859x_7 \end{aligned}$$

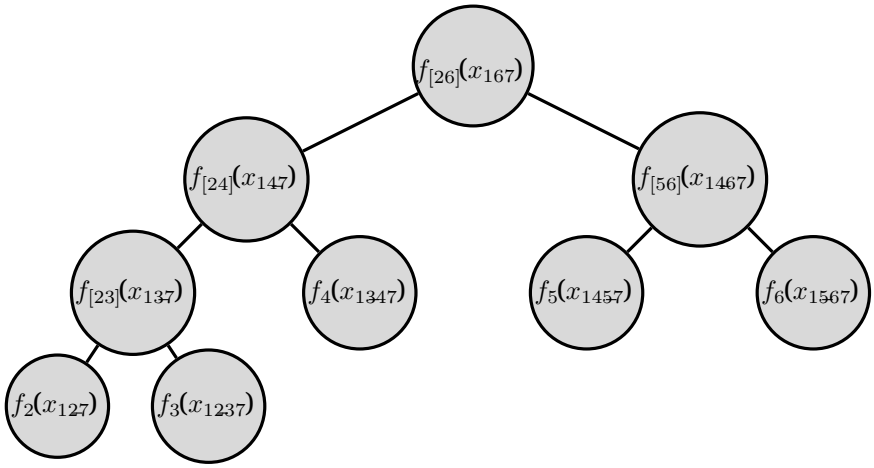

图2：描述多项式 $f_{[i+1,j]} \in \mathbb{F}_p[x_1, x_i, x_j, x_n]$ 迭代成对计算过程的二叉树。首个结式的具体说明见例1。记号 $f(x_{abc})$ 表示多项式 f 含有变量 x_a，x_b，x_c，且变量 x_b 将在下一步中消去。

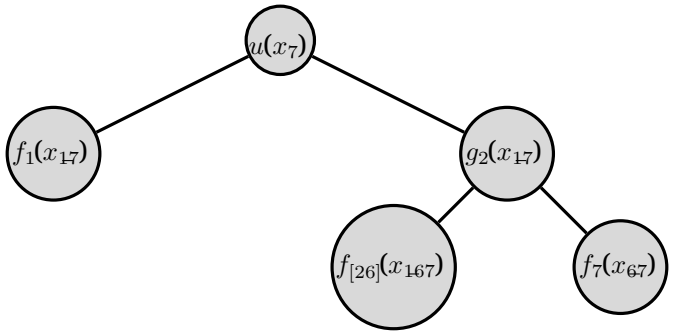

图3：当 $n = 7$ 时算法的最后步骤。请注意，算法只有这部分依赖于 f_7，因而也依赖于参数 t。多项式 u 的解对应的值 x_n 即为所需输出。

消去 x_5 后第一对多项式的结式多项式：

$$\begin{aligned}\mathrm{Res}(f_6,f_5;x_5) &= 933x_1^2x_4^4x_6+388x_1^2x_4^4x_7+1494x_1^2x_4^3x_6x_7+936x_1x_4^4x_6x_7\\ &+431x_1x_4^4x_7^2+1322x_1x_4^3x_6x_7^2+801x_4^4x_6x_7^2+49x_4^4x_7^3\\ &+1327x_4^3x_6x_7^3+686x_1^2x_4^3+646x_1x_4^4+494x_1^2x_4^2x_6\\ &+462x_1x_4^3x_6+1522x_4^4x_6+718x_1^2x_4^2x_7+698x_1x_4^3x_7\\ &+679x_4^4x_7+29x_1^2x_4x_6x_7+476x_1x_4^2x_6x_7+330x_4^3x_6x_7\\ &+460x_1x_4^2x_7^2+158x_4^3x_7^2+201x_1x_4x_6x_7^2+1286x_4^2x_6x_7^2\\ &+1229x_4^2x_7^3+196x_4x_6x_7^3+837x_1^2x_4+693x_1x_4^2\\ &+1519x_4^3+933x_1^2x_6+1061x_1x_4x_6+6x_4^2x_6\\ &+388x_1^2x_7+825x_1x_4x_7+495x_4^2x_7+936x_1x_6x_7\\ &+1193x_4x_6x_7+431x_1x_7^2+1365x_4x_7^2+801x_6x_7^2\\ &+49x_7^3+646x_1+4x_4+1522x_6+679x_7\end{aligned}$$

第二对多项式：

$$\begin{aligned}f_3 &= 1171x_1x_2^2+704x_1x_2x_3+x_2^2x_3+685x_2^2x_7+153x_2x_3x_7\\ &+352x_1+2x_2+1522x_3+838x_7\\ f_4 &= 1288x_1x_3^2+470x_1x_3x_4+x_3^2x_4+1409x_3^2x_7+228x_3x_4x_7\\ &+235x_1+2x_3+1522x_4+114x_7\end{aligned}$$

消去 x_3 后第二个结式多项式：

$$\begin{aligned}\mathrm{Res}(f_4,f_3;x_3) &= 1497x_1^2x_2^4x_4+1003x_1^2x_2^4x_7+557x_1^2x_2^3x_4x_7+1014x_1x_2^4x_4x_7\\ &+365x_1x_2^4x_7^2+63x_1x_2^3x_4x_7^2+830x_2^4x_4x_7^2+1256x_2^4x_7^3\\ &+1068x_2^3x_4x_7^3+1419x_1^2x_2^3+931x_1x_2^4+156x_1^2x_2^2x_4\\ &+845x_1x_2^3x_4+1522x_2^4x_4+74x_1^2x_2^2x_7+1010x_1x_2^3x_7\\ &+9x_2^4x_7+966x_1^2x_2x_4x_7+8x_1x_2^2x_4x_7+1487x_2^3x_4x_7\\ &+856x_1x_2^2x_7^2+274x_2^3x_7^2+1460x_1x_2x_4x_7^2+1112x_2^2x_4x_7^2\\ &+79x_2^2x_7^3+455x_2x_4x_7^3+104x_1^2x_2+506x_1x_2^2+1519x_2^3\\ &+1497x_1^2x_4+678x_1x_2x_4+6x_2^2x_4+1003x_1^2x_7\\ &+513x_1x_2x_7+1469x_2^2x_7+1014x_1x_4x_7+36x_2x_4x_7\\ &+365x_1x_7^2+1249x_2x_7^2+830x_4x_7^2+1256x_7^3\\ &+931x_1+4x_2+1522x_4+9x_7\end{aligned}$$

按照图 *2* 所示计算完其余两两结式后，最上层结式 $f_{[26]}$ 中剩下变量 x_1、x_6、x_7。现在，我们可以将其与 f_7 合并并消去 x_6。最后，我们将其与 f_1 合并（在初始阶段已消去其中的 x_0），从而得到单变量多项式 u（x_7）。

4.3 ResultantSolver 算法

本节包含一个分为两部分的伪代码式算法：第（1）部分和第（2）部分。第（1）部分计算所有多项式 f_2、.....、f_{n-1} 的结式，最终得到一个多项式 g_3。完成这部分后，剩下由三个多项式方程 $f_1 = 0$、$g_3 = 0$ 和 $f_n = 0$ 组成的方程组，涉及变量 x_1、x_{n-1} 和 x_n，且只有 f_n 包含系数 t。第（2）部分通过计算最后两个结式 $g_2 = \text{Res}(f_n、g_3; x_{n-1})$ 和 $u(x_n) = \text{Res}(f_1、g_2; x_1)$，推导出特定 t 值对应的最终单变量多项式 $u(x_n)$。最后，使用求根算法求解 $u(x_n) = 0$。

算法 1 ResultantSolver：
描述求解（2）过程的伪算法

要求： 方程（2）的 GMV 方程组
确保： 满足方程组的 x_n 取值。
第（*1*）部分：预计算
使用 f_0 消去 x_0，按照第 2.1 节所述，将 f_1' 替换为 f_1。
$S \leftarrow \{f_2, \ldots, f_{n-1}\}$
对 $\{2, \ldots, n-2\}$ 中按 4.2 节给定顺序排列的每个变量 i **执行**
选择两个多项式 f，$g \in S$，它们的变量集都包含 x_i
使用 (4) 将 S 中的 f 和 g 替换为 $\text{Res}(f, g; x_i)$ // 消去变量 x_i
结束循环
S 中的单个剩余多项式
第（*2*）部分：（可对 t 的不同取值迭代）
对于 t 的多个取值**执行**
令 t 属于 f_n
$g_2 \leftarrow \text{Res}(f_n, g_3; x_{n-1})$
$u \leftarrow \text{Res}(f_1, g_2; x_1)$
求 $u(x_n)$ 在 $\mathbb{F}$ 中的根 $_p$ 使用 Berlekamp–Rabin 算法。
结束循环

4.3.1 时间复杂度

根据猜想 1，我们预计复杂度中含有一个关于 n 的指数因子。此外，由于需要处理模 p（一个非常大的数）的整数，我们还预计会出现一个因子 $\log_2(p)$。因此，可以假设部分 (1)+(2) 的总体执行时间形式为 $\mathcal{O}(2^{un}\log_2(p))$，其中 $u > 0$。实验依据（见表 3）表明，在许多情况下，我们可能有 $u < 1$。

4.3.2 空间复杂度

很难估算最佳内存使用量。甚至无法确定最后一个以次数 $3(2^n - 1)$ 结束的结式是最繁重的步骤；事实上，第（1）部分中的最后一个结式可能更大。因此，尽管我们观察到

n	7	8	9	10	11	12
Magma	0.28	1.95	13.91	100.33	781.38	6246.79
第 1 部分	0.002	0.007	0.019	0.049	0.090	0.312
第 2 部分	0.005	0.010	0.019	0.032	0.070	0.152

表 1：求解 GMV 多项式系统 (2) 在 n 取值递增时的执行时间（以秒为单位）。“Magma”给出 Magma V2.27-1 中实现的 Variety 方法的总时间开销。第 1 部分给出算法 1 第 1 部分（打包稀疏结式）的总时间开销，第 2 部分给出为 t 的每个取值计算解所需的额外时间。在两种情况下，测试均在同一硬件平台上进行，即一台配备 192GB RAM 的工作站，但未使用任何并行化。所有测试均使用在域 $\mathbb{F}_p$ 上随机生成的示例进行，并带有 $p = 8380417$。

内存是该算法的瓶颈，但很难分析在动态设置下减少内存的所有方案；这一点也在第 4.5 节中进行了简要讨论。

4.4 与现有方法的比较

下文中，我们将本方法的效率与求解（一般）多项式系统的当前最先进方法进行比较。后者以计算机代数系统 Magma V2.27-1 中的 Variety(·) 功能为代表。对于第 2 节所述与 GMV 多项式系统相关联的理想，该 Variety 方法由三个步骤组成。首先，它使用 F_4-算法 [3] 计算反向字典序下的 Gr¨obner 基。第二步，利用 FGLM 算法 [2] 的一个变体，将该 Gr¨obner 基变换为字典序下的 Gr¨obner 基。这第二个基包含理想中的一个单变量多项式。第三步亦即最后一步求出该单变量多项式的根，并恢复相关解的其余部分。在我们的测试中，运行时间主要由前两个步骤主导。

在表 1 中，我们比较了 Magma 中 Variety 方法与前文所述结式方法在系数随机生成的 GMV 多项式系统上的运行时间 a_i，b_i，t。表 1 中的运行时间是在同一台配备 192GB RAM 的计算机上运行程序得到的。对于参考用的 Gr¨obner 基算法，我们观察到，当 p 固定时，执行时间大致增长为原来的 $\approx \mathcal{O}(2^{3n})$ 倍。如果底层理想处于形状位置，那么这与命题 4 一致，因为我们预计 FGLM 步骤的增长倍数为 $\mathcal{O}(2^{\omega n})$，其中线性代数常数的实际取值范围为 $2.81 \leq \omega \leq 3$。

表 3 进一步给出了在规模更大的 (2) 实例上运行算法 1 时所测得复杂度的详细信息。这些实例是在配备 16GB 内存和 Apple M2 处理器的 MacBook Air 上运行的，因此其执行时间比工作站更短。在附录 A 中，我们还展示了

n	13	14	15	16	17	18
第 1 部分	0.459	2.425	5.075	10.333	23.678	102.566
第 2 部分	0.330	0.731	1.615	3.534	7.660	16.471

表 2：求解 n 的 GMV 多项式系统 (2) 时的执行时间（单位：秒）。第 1 部分给出了算法 1 第 1 部分的总时间开销，第 2 部分给出了算法 1 第 2 部分中每次迭代所需的时间。测试在配备 16GB RAM 和 Apple M2 处理器的 MacBook Air 上进行。所有测试均使用在域 $\mathbb{F}_p$ with $p = 8380417$ 上随机生成的实例完成。

我们程序的完整打印输出，包括 [5]中所给示例系统的实际解。

表 3 比较了求解部分示例系统时的计算时间。

GMV 示例	n	$\log_2(p)$	dup	部分(1)	部分(2)	ret
A	13	18	24573	0.453		
					0.262	1.0
					0.263	1.0
G	17	22	393213	23.577		
					7.530	≈ 81
					7.506	≈ 81
H	17	73	393213	72.129		
					42.087	≈ 160
					53.246	≈ 175

表 3：算法 1 第 (1) 部分和第 (2) 部分的执行时间，单位为秒。此处，dup $=$ 最终单变量多项式的次数。在每种情况下，dup 等于$3 \cdot (2^n - 1)$。ret $=$ 第 (1) 部分+(2) 的相对执行时间，经缩放后使例A的总时间为 1.0。可以看到，情形 H 与 G 的总执行时间相差接近 2 倍，并且情形 G 与 A 的总执行时间相差一个因子 ≈ 81 ，即3^{17-13}。更多细节请参见附录 A。

4.5 一些潜在改进

为了加快处理速度，可以使用一些我们迄今尚未采用的技巧。

- 并行化：

– 第 1 部分的某些操作可以并行化。特别是，与图2中不同子树对应的进程可以并行执行。– 在更细粒度的层面上，式（4）中的结式计算由相互独立的部分组成，适合进行并行化。

• 优化二元结果树中节点对的选择（参见表 2），以便在接近树顶时避免出现“繁重”的节点。换言之，二叉树左右子树的深度应尽可能接近。

• 适应硬件环境：在我们的实验中，可用 RAM 的容量是限制我们能够处理哪些实例的瓶颈。我们可以更频繁地释放内存，但代价是必须多次计算相同的中间值。这是一种时间/内存权衡，而确定最优策略可能并非易事。

5 总结

我们提出了一种求解 GMV 方程组的方法，其速度显著快于暴力搜索或标准的 Gr¨obner 基方法所能达到的速度。该方法利用了结式 理论以及方程组的稀疏性，并且本身支持并行化（目前我们尚未利用这一点）。

尽管目前距离 $n = 521$ 的目标还很遥远，但我们认为仍有改进空间，以便处理更多示例案例。具体而言，充分发挥并行化潜力并优化内存使用，应能进一步提升性能。不过，我们也认为，任何从系统生成的理想中挑出一个单变量多项式的代数算法，其复杂度都必须达到 $\mathcal{O}(2^n)$。这意味着，如果 p 应为一个521位素数，那么实现 $n \approx \log_2(p)$便遥不可及。如果能够放宽 n与 p之间的关系，那么即使 p 是一个521位素数，也完全有可能在 n 较小（例如 $n \approx 20$）时求解该系统。

参考文献

[1] A. Bariant、A. Boeuf、P. Briaud、M. Hostettler、H. Raddum 与 M. Øygarden。针对算术化导向原语的改进结果式攻击。载于*CRYPTO25, LNCS 16004*，第335–367页，2025年。[2] J.-C. Faugere、P. Gianni、D. Lazard 与 T. Mora。通过改变序高效计算零维 Gr¨obner 基。《符号计算杂志》，16(4):329–344，1993年。[3] J.-C. Faug´ere。一种计算 Gr¨obner 基的新高效算法（F_4）。《纯粹与应用代数杂志》，139(1-3):61–88，1999年。

[4] GMV。GMV 通过解决一项国际密码学挑战认可数学创新，2025年。 https://www.gmv.com/en-es/communication/news/gmv-recognizes-mathematical-innovation-solving-international-cryptographic.

[5]GMV 和 Trampoline。《有限域迷宫中的多项式》，2025。 https://www.unipv.news/sites/magazine/files/2025-04/2025_Polynomials%20in%20the%20Maze%20of%20Finite%20Fields.pdf。

[6] J. Von Zur Gathen 和 J. Gerhard。《现代计算机代数》。剑桥大学出版社，2003。

[7] H.-S. Yang、Q.-X. Zheng、J. Yang、Q.-F. Liu 和 D. Tang。基于结式的面向算术算法新安全性评估方法。载于*Asiacrypt 24，LNCS 15490*，第 457–489 页，2024。

求解示例系统的打印输出

作为参考，我们在此列出使用我们的实现运行算法 1 时，针对 [5] 中给出的示例系统生成的打印输出。这些运行是在配备 16GB 内存和 Apple M2 处理器的 MacBook Air 上完成的，并非用于表 1 中计时测试的同一台机器。

A.1 示意性玩具示例

系统已初始化：n=7，p 为 11 比特，稀疏结式打包已完成，用时 0.009 秒。t=1268 的次数为 381 的单变量多项式的根：x7 = 180 已在 0.005 秒内找到解

A.2 情况 A

系统已初始化：n=13，p 为 18 比特，稀疏结式打包耗时 0.453 秒。g2 有 24573 项。t=28555 对应的 24573 次单变量多项式的根：x13 = 176007。在 0.263 秒内找到解。g2 有 24573 项。t=91766 对应的 24573 次单变量多项式的根：x13 = 98529x13 = 100260x13 = 39269。在 0.262 秒内找到解。g2 有 24573 项

t=99194 对应的 24573 次单变量多项式的根：x13 = 129426x13 = 192514。在 0.263 秒内找到解。g2 有 24573 项。t=145052 对应的 24573 次单变量多项式的根：x13 = 76075x13 = 175914x13 = 112372x13 = 179016。在 0.263 秒内找到解。g2 有 24573 项。t=85098 对应的 24573 次单变量多项式的根：x13 = 25091。在 0.263 秒内找到解

A.3 案例 B

系统已初始化：n=13，p 有 18 比特，稀疏结式打包已在 0.456 秒内完成。g2 有 24573 项。t=2678 对应的次数为 24573 的单变量多项式的根：x13 = 211x13 = 19582，在 0.262 秒内找到解。g2 有 24573 项。t=142207 对应的次数为 24573 的单变量多项式的根：x13 = 1091，在 0.262 秒内找到解。g2 有 24573 项。t=179873 对应的次数为 24573 的单变量多项式的根：x13 = 165690x13 = 136856x13 = 31030，在 0.262 秒内找到解。g2 有 24573 项。t=18905 对应的次数为 24573 的单变量多项式的根：x13 = 80081，在 0.262 秒内找到解。g2 有 24573 项。t=59939 对应的次数为 24573 的单变量多项式的根：x13 = 123952x13 = 64141，在 0.263 秒内找到解

A.4 案例 C

系统已初始化：n=13，p 有 18 比特，稀疏结式打包已在 0.451 秒内完成。g2 有 24573 项。t=63664 对应的次数为 24573 的单变量多项式的根：x13 = 141159x13 = 129533x13 = 19991，在 0.263 秒内找到解。g2 有 24573 项。t=96122 对应的次数为 24573 的单变量多项式的根：x13 = 159594x13 = 191455x13 = 60057，在 0.265 秒内找到解。g2 有 24573 项。t=141701 对应的次数为 24573 的单变量多项式的根：x13 = 141693，在 0.263 秒内找到解。g2 有 24573 项。t=35444 对应的次数为 24573 的单变量多项式的根：x13 = 38135x13 = 173393x13 = 30357，在 0.265 秒内找到解。g2 有 24573 项。t=120184 对应的次数为 24573 的单变量多项式的根：x13 = 189726x13 = 28259x13 = 69279，在 0.264 秒内找到解

求解这两个示例系统的 $n = 17$，清楚地表明：如果保持 n 不变，求解复杂度仅随 $\log_2(p)$ 线性增长。

A.5 案例 G

系统已初始化：n=17，p 有 22 比特，打包稀疏结式已完成，耗时 23.577 秒。g2 有 393213 项。一元多项式的根，次数为 393213，t=3162200：x17 = 2905037x17 = 423964已找到解，耗时 7.530 秒。g2 有 393213 项。一元多项式的根，次数为 393213，t=1948684：

x17 = 414099已找到解，耗时 7.506 秒。g2 有 393213 项。一元多项式的根，次数为 393213，t=2832793：x17 = 2773074x17 = 344215x17 = 2158164已找到解，耗时 7.516 秒。g2 有 393213 项。一元多项式的根，次数为 393213，t=452267：x17 = 3119706已找到解，耗时 7.520 秒。g2 有 393213 项。一元多项式的根，次数为 393213，t=1114540：x17 = 3462664已找到解，耗时 7.501 秒

A.6 情形 H

系统已初始化：n=17，p 为 73 比特。稀疏结式打包耗时 72.129 秒。g2 有 393213 项。t=1058371656331684574765 对应的次数为 393213 的单变量多项式的根：x17 = 319085019487445514268 6x17 = 44273923048250855801 58x17 = 217347983333160095478，42.087 秒后找到解。g2 有 393213 项。t=21875805440386776668 对应的次数为 393213 的单变量多项式的根：x17 = 1868566290398595250336，53.246 秒后找到解。g2 有 393213 项。t=3843998821665876069844 对应的次数为 393213 的单变量多项式的根：x17 = 1298918500212281416535x17 = 3180019097753843068048x17 = 4714514066403504455654，62.847 秒后找到解。g2 有 393213 项。t=48043857385386667728 对应的次数为 393213 的单变量多项式的根：x17 = 3085726008969165404256，68.707 秒后找到解。g2 有 393213 项。t=104940936778785605637 对应的次数为 393213 的单变量多项式的根：x17 = 417928365705601228433 0x17 = 3129644745497065872124x17 = 2702577589972463891281

```
x17 = 1708563171310648447784x
17 = 5180557299946937162500在 77.882
秒内找到解
```