求解稀疏多元多项式插值问题的分治算法

邓国强 唐 敏 梁状昌

(桂林电子科技大学广西密码学与信息安全重点实验室 广西 桂林 541004)

摘 要 稀疏多元多项式插值被广泛应用在科学和工程领域,目标是利用多项式的稀疏结构及其给定的离散信息恢复目标多项式。目前的主流方法在目标多项式规模较大时均表现出较高的时间复杂度,因其所需的代数操作的规模及个数与多项式的项数和次数相关。鉴于此,提出了一种求解稀疏多元多项式插值问题的有限域上的分治算法,其基本策略是视多项式中的一个变元为主元,其系数为关于其他变元的多元多项式,从而将原问题分解为一系列单变元多项式插值及规模远小于原问题的一系列子多元多项式插值问题,合并这些子多元多项式即得到原问题的解。为实现稀疏多元多项式插值分治算法,设计了4个子算法:基于提前终止策略的单变元多项式插值算法、已知次数的单变元多项式插值算法、多项式项数判定的Hankle矩阵行列式检测法、已知项数的Ben-Or/Tiwari算法。对新算法与Zippel算法、Ben-Or/Tiwari算法、Javadi/Monagan算法进行了数值实验比较,结果表明所提算法在运行时间上有较大的改进。实验数据充分说明:提前终止策略的运用,消除了必须给定目标多项式的项数界和次数界的限制;分治策略的运用,将大量高阶的代数运算分解为低阶问题,从而有效地解决了大规模多元多项式插值问题的时间性能瓶颈。

关键词 稀疏多元多项式插值,分治算法,提前终止策略,Hankel矩阵

中图法分类号 TP301.6,O241.3 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.05.046

Divide-and-Conquer Algorithm for Sparse Polynomial Interpolation

DENG Guo-qiang TANG Min LIANG Zhuang-chang

(Guangxi Key Laboratory of Cryptography and Information Security,Guilin University of Electrical Technology,Guilin,Guangxi 541004,China)

Abstract Sparse interpolation is widely used in quite different applications and areas of science and engineering. Its goal is to recover the goal polynomial by taking advantage of the sparse structure of the polynomial and given discrete values. For polynomials with the large size,the current methods show high time complexity,because the size and the number of algebra operations are related to the number of terms and the total degree of the goal polynomials. For this reason,this paper presented a divide-and-conquer algorithm for sparse polynomial interpolation over finite fields. The basic strategy is to choose one of variables as the main variable and the coefficients are multivariate polynomials in other variables. In this way,the original polynomial interpolation is divided into a list of univariate polynomial interpolations and a list of sub-polynomials with smaller size. The solution of the original problem is to merge these sub-polynomials. To implement the divide-and-conquer strategy for the sparse polynomial interpolation,this paper designed four sub-algorithms:univariate polynomial interpolation based on early termination strategy,univariate polynomial interpolation with a prior knowledge of the total degrees of the polynomial,the determination of the number of terms of the polynomial via Hankle matrix determinant,and Ben-Or/Tiwari's algorithm with an upper bound of the number of terms. In numerical experiments,the performance of the new algorithm is compared with that of Zippel's algorithm,Ben-Or/Tiwari's algorithm and Javadi/Monagan's algorithm. Extensive experiments show that the new algorithm is much faster than other three algorithms. The experimental data demonstrate that the use of divide-and-conquer and early termination strategy not only eliminates some priori knowledge of the total degree and the number of terms of the goal polynomials,but also decomposes a large number of higher order algebra operations into smaller ones. Therefore,the bottleneck of the large-scale multivariate polynomial interpolation problems is effectively solved.

Keywords Sparse multivariate polynomial interpolation,Divide-and-conquer algorithm,Early termination strategy,Hankel matrix

到稿日期:2018-12-06 返修日期:2019-02-07 本文受广西科技基地和人才专项(桂科AD18281024),广西高校中青年教师基础能力提升项目(2019KY0210,2018KY0210),国家级大学生创新训练计划项目(201810595024)资助。

邓国强(1979-),男,硕士,讲师,主要研究方向为进化计算优化方法,E-mail:d9801242@126.com;**唐 敏**(1980-),女,博士,副教授,主要研究方向为计算机代数,E-mail:dengtangmin@126.com(通信作者);**梁状昌**(1996-),男,主要研究方向为数值优化算法。

1　引言

稀疏插值被广泛应用在数学和数值领域[1-4]及科学和工程领域[5-7]，比如有理近似[1-2]、符号计算[3-4]、信号处理[5-6]、图像处理、压缩感知、指数分析[7]、广义特征值问题等。假设目标函数 $f(x)=\alpha_1 x^{100}+\alpha_2$，传统的插值算法需要 101 个插值点，而稀疏插值仅需 4 个插值点。当目标函数为多元多项式时，传统插值算法为指数复杂度，而稀疏插值可实现多项式复杂度 $O(d^n)$。稀疏多元多项式插值的研究目标即为使用较少的插值点，在较低的时间复杂度算法下重构目标函数。

稀疏多元多项式插值问题定义为：令 $P=\sum_{i=1}^{t} C_i M_i(x_1, x_2, \cdots, x_n)$ 是用黑盒 $B:\mathcal{R}^n\to\mathcal{R}$ 表示的多元多项式，其中 $C_i\in\mathcal{R}/\{0\}$，$M_i=x_1^{e_{i1}}x_2^{e_{i2}}\cdots x_n^{e_{in}}$，$e_{il}\in Z$ 是 $t(t\ll d^n)$ 个互不相同的单项式，d 是 P 的全次数。黑盒是一个具有如下功能的装置：对于任意输入 n 元组 $(\alpha_1,\alpha_2,\cdots,\alpha_n)$，输出 $P(\alpha_1,\alpha_2,\cdots,\alpha_n)$。稀疏多元多项式插值问题为利用 P 的稀疏结构及黑盒给出的离散信息，确定 C_i 及 e_{il}，恢复黑盒表示的多元多项式 P 的函数形式。

图 1 给出了包含 2 个变元的多项式的黑盒插值问题的示例。若给定若干输入，如(1,1)，(2,3)，(4,9)，…，则黑盒输出 P 在这些点处的函数值，如将此绘制在图 2(a)所示的三维空间中，即获得一系列离散点信息；黑盒插值问题要求根据这些离散信息恢复黑盒多项式的函数形式，得到图 2(b)所示的空间曲面。在实际问题中即表示给定若干个信号的输入、输出，恢复信号的函数表达式(多元多项式形式)。

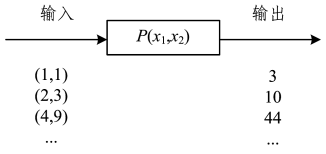

图 1　黑盒插值问题示例

Fig. 1　Example of black box interpolation

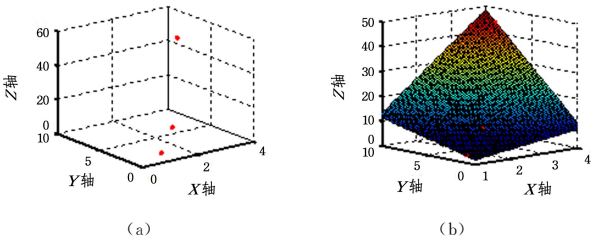

图 2　黑盒插值问题图例

Fig. 2　Diagram of black box interpolation

不考虑多项式的稀疏结构，多元多项式插值所需的赋值点个数为 $(d+1)^n$，即最坏情形下是指数级计算复杂度。1979 年，Zippel 提出了高概率稀疏多元多项式插值算法[8]，其复杂度为 $O(ndt^3)$，它是第一个多项式时间复杂度算法。但是逐个变元地顺序插值不利于并行化的实现，同时大量线性方程组的求解也导致多项式规模较大、时间效率较低。与 Zippel 概率性算法不同，Ben-Or 和 Tiwari 在 1988 年给出了一种确定性的算法[9]，它使用了一种特殊的编码格式，时间复杂度为 $O(t^2(\log^2 t+\log nd))$。该算法一次可对多个变元同时插值，由于插值点选用素数的幂次，对于变元个数较多的多项式，编码长度较大，计算复杂度较高。关于 Zippel 和 Ben-Or/Tiwari 这两种算法的改进和变体可参考文献[10-14]。为了处理有限域上的代数问题，Javadi 和 Monagan 在 2010 年提出了有限域上的并行稀疏插值算法[15]，它是对 Ben-Or/Tiwari 算法进行的改进，通过增加额外的插值点，将插值多项式从特征为 0 的环扩展到特征为 p(素数)的环；Arnold 等学者随后提出了适用于变元个数较多的有限域上快速多元多项式稀疏插值算法[16-17]。同期，支丽红等在计算稀疏性及数值稀疏插值所需的最少插值点的估计方面进行了相关研究[18-19]。最新的稀疏插值算法是 Huang 在 2018 年给出的高概率递归算法[20]，递归层次较多和非有限域上的计算使得该算法仍有改进的空间。以上提到的文献中(除文献[20])，为了完成稀疏多元多项式插值，算法均需进行与目标多项式同等规模的代数运算，包括线性方程组求解、行列式计算、多项式求根、素数分解等，因此当目标多项式规模较大时，计算效率较低。

借鉴文献[9,20]，本文提出了一种有限域上的分治算法来求解稀疏多元多项式插值问题。基本策略是将变元 x_1 视为主元并合并同类项，则 x_1 的系数为关于 $x_2,x_3,\cdots,x_n$ 的多元多项式。利用牛顿插值或拉格朗日插值获得在特定点处的一系列关于变元 x_1 的单变元多项式，再利用 Ben-Or/Tiwari 算法恢复 x_1 的各个系数多项式，所有运算均在有限域上进行，对中间表达式的严重膨胀进行了有效控制，最后利用有理数恢复算法得到原问题的解。在此策略下，原插值问题被分解为若干个单变元多项式插值及规模远小于原多项式的一系列多元多项式插值问题，最终将这些多元多项式作为 x_1 的系数多项式，对其合并后得到原问题的解。

本文第 2 节介绍基本设计策略及思想，以及稀疏多元多项式插值分治算法的流程；第 3 节给出用于实现分治算法的 4 个子算法和实例，并分析其时间复杂度；第 4 节给出数值实验，将分治算法与 3 种流行算法进行比较；最后总结全文。

2　基本设计策略及思想

2.1　设计思想

稀疏多元多项式插值的目标为使用尽可能少的插值点和多项式时间复杂度较低的算法恢复目标多项式。

分治算法求解稀疏多元多项式插值问题的设计思想来源于以下 3 点。

(1)通过观察并测试验证发现，若以多项式中一个变元(例如 x_1)为主元，合并同类项后，该主元的系数多项式(关于变元 $x_2,x_3,\cdots,x_n$ 的多元多项式)的规模远小于多项式 P。例如，令：

$$P=-46x_1^4x_2^6-69x_1^7x_2^2-16x_1^5x_2^2x_3+59x_1^4x_2x_3^3+30x_1^5x_2-64x_1^2x_2^4+89x_2^4x_3^2-35x_1^3x_3+97x_1x_3^3+21x_1^3$$

其中，变元个数为 3，项数为 10，全次数为 10。将 P 以 x_1 为主元，合并同类项，重写为：

$$P=-69x_2^2x_1^7+(-16x_2^2x_3+30x_2)x_1^5+(-46x_2^6+59x_2x_3^3)x_1^4+(-35x_3+21)x_1^3-64x_2^4x_1^2+97x_3^3x_1+89x_2^4x_3^2$$

那么，x_1 的系数多项式有 7 个项数至多为 2 的多元多项式。

表 1 列出了一组变元个数、项数、次数互异的多元多项式 P，其中 n 为变元个数，d 为 P 的全次数，t 为 P 的项数，τ 为分解后 x_1 的系数多项式的最大项数。通过对比表 1 中第 3 列和第 4 列发现，重写 P 后系数多项式的规模远小于原多项式的规模。该示例说明多项式 P 的插值问题可分解为若干个规模远小于 P 的插值问题。

表 1 系数多项式规模示例

Table 1 Size of coefficient polynomials

n	d	t	τ
3	20	20	3
5	20	20	4
8	20	20	5
3	30	30	5
5	30	30	4
8	30	30	6

(2)单变元多项式插值的计算复杂度远低于多元多项式的计算复杂度。为了使用 Ben-Or/Tiwari 算法对 x_1 的系数多项式插值，需要获得在形如 $(2^i,3^i,\cdots,p_{n-1}^i)(i=0,1,\cdots,k)$ 的插值点处的取值，利用牛顿或拉格朗日插值获得此信息的时间复杂度为 $O(k\log_2^2k)$。该操作仅为算术运算，不包含方程组求解、求根等复杂的代数运算。

(3)使用 Ben-Or/Tiwari 算法插值多元多项式，所需插值点的个数恰为 $2t$，所需求解的线性方程组的规模为 $t\times t$，时间复杂度是 $O(t^2(\log^2t+\log nd)$，因而项数越少则效率越高。

2.2 分治算法及流程

基于以上 3 点观察，本文设计了一种分治算法来求解稀疏多元多项式插值问题。令黑盒多元多项式为：

$$P(\boldsymbol{X})=\sum_{i=1}^{t}C_iM_i(x_1,x_2,\cdots,x_n)$$

将 P 写成以变元 x_1 为主元的多项式，其中 x_1 的系数是关于变元 $x_2,x_3,\cdots,x_n$ 的多元多项式 f_i，有：

$$P(\boldsymbol{X})=\sum_{i=1}^{s}f_i(x_2,x_3,\cdots,x_n)x_1^{e_i}$$

令 p_i 为第 i 个素数，利用单变元牛顿或拉格朗日插值获得关于变元 x_1 的多项式序列：

$$P_0=\sum_{i=1}^{s}f_i(p_1^0,p_2^0,\cdots,p_{n-1}^0)x_1^{e_i}$$

$$P_1=\sum_{i=1}^{s}f_i(p_1^1,p_2^1,\cdots,p_{n-1}^1)x_1^{e_i}$$

$$\vdots$$

$$P_{2\tau-1}=\sum_{i=1}^{s}f_i(p_1^{2\tau-1},p_2^{2\tau-1},\cdots,p_{n-1}^{2\tau-1})x_1^{e_i}$$

其中，$\tau=\max\{\mathrm{term}(f_i)\}$，$i=1,2,\cdots,s$，$\mathrm{term}(f_i)$ 表示 f_i 的项数。

至此，将目标多项式 P 的插值问题分解为了 s 个项数至多为 τ 的多项式 f_i 的插值，使用 Ben-Or/Tiwari 算法插值恢复 x_1 的各个系数多项式 f_i，合并起来即为原多项式 P。算法流程如图 3 所示，其中 $\boldsymbol{p}^{(j)}=(p_1^j,p_2^j,\cdots,p_{n-1}^j)$。

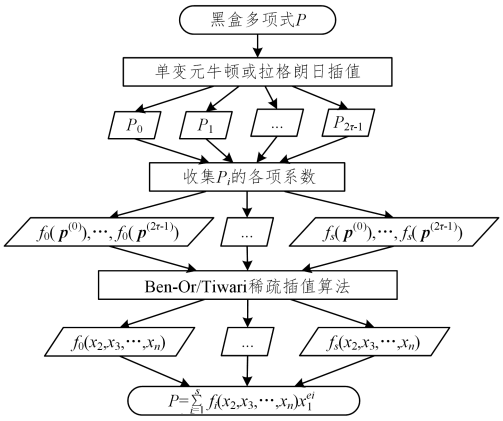

图 3 稀疏多元多项式插值分治算法的流程图

Fig. 3 Flow chart of divide-and-conquer algorithm for sparse interpolation

3 稀疏多元多项式插值问题的分治算法

根据第 2 节的思想，本节给出稀疏多元多项式插值问题的分治算法的详细过程。首先介绍分治算法中包含的 4 个子算法(见 3.1—3.4 节)，然后给出完整的分治算法及时间复杂度分析(见 3.5 节)，最后给出一个具体实例(见 3.6 节)。

根据分治算法流程图(见图 3)，首先需要使用单变元的牛顿或拉格朗日插值计算关于 x_1 的单变元多项式序列 P_0，$P_1,\cdots,P_{2\tau-1}$。由于 x_1 次数未知，3.1 节给出了基于提前终止策略的单变元多项式插值算法(算法 1)，求出 P_0 的同时获得了 x_1 的次数；然后通过 3.2 节给出的已知次数的单变元多项式算法完成 $P_1,P_2,\cdots,P_{2\tau-1}$ 的计算(算法 2)。

接下来收集 $P_1,P_2,\cdots,P_{2\tau-1}$ 的各项系数并通过 Ben-Or/Tiwari 算法恢复系数多项式 $f_1,f_2,\cdots,f_s$。由于 Ben-Or/Tiwari 算法要求给定目标多项式的项数界，因此在 3.3 节中设计了判定多项式项数的行列式检测法(算法 3)用于求出 f_1，$f_2,\cdots,f_s$ 的准确项数；最终使用 3.4 节的 Ben-Or/Tiwari 算法(算法 4)恢复 x_1 的各个系数多项式 f_i，将其合并起来即为原多项式 P。

3.1 基于提前终止(Early Termination)策略的单变元多项式插值算法

稀疏多元多项式插值的分治算法需要获得一系列在特定点处关于变元 x_1 的单项式，由于 x_1 的最高次数未知，需要给出一个基于提前终止策略的单变元多项式插值算法。该设计思想来源于定理：给定单变元多项式 $P(x)=a_0+a_1x+\cdots+a_dx^d$ 的 $k(k>d)$ 个互异点处的函数值 $y_0=P(x_i)(i=0,1,\cdots,k)$，插值多项式存在且唯一。逐渐增加插值点，构造插值多项式，检测新产生的多项式是否与前一个相同，如果结果是肯定的，即可获得 $P(x)$。

算法 1 基于提前终止策略的单变元多项式插值算法(ETUP)

输入：包含 n 个变元的黑盒多项式 $P(\mathbf{X})=\sum_{i=1}^{s}f_i(x_2,x_3,\cdots,x_n)x_1^{e_i}$

输出：P 在点 $(1,1,\cdots,1)$ 处关于变元 x_1 的多项式 $P_0=\sum_{i=1}^{s}f_i(1,1,\cdots,1)x_1^{e_i}$，以及 x_1 的最高次数 $d=\max\{e_i\}$，$i=1,2,\cdots,s$

Step 1　设置初值 i：=2，P_{00}：=0。

Step 2　选择插值点 $u_j=(j,1,1,\cdots,1)$，j=1，2，…，i，令 $v_j=P(u_j)$，使用牛顿插值或拉格朗日插值计算由点$[(u_1,v_1),\cdots,(u_i,v_i)]$确定的插值多项式 P_{0i}，如果 $P_{0i}=P_{0,i-1}$，返回 P_0：=P_{0i}和 x_1 的最高次数 d；否则 i：=i+1，转到 Step 2。

3.2　已知次数的单变元多项式插值算法

若已知单变元多项式 $P_j(j=1,2,\cdots)$的最高次数 d，那么可直接构造关于变元 x_1 的多项式 $P_j=\sum_{i=1}^{s} f_i(p_1^j,p_2^j,\cdots,p_{n-1}^j)x_1^{e_i}$，其中 p_k 表示第 k 个素数。

算法 2　已知次数的单变元多项式插值算法(UP)

输入：p_k 的幂次 j，变元 x_1 在 P_j 中的最高次数 d

输出：$P_j=\sum_{i=1}^{s} f_i(p_1^j,p_2^j,\cdots,p_{n-1}^j)x_1^{e_i}$

Step 1　选择插值点 $u_i=(2^j,3^j,\cdots,p_{n-1}^j)$，i=0，1，…，d，令 $v_i=P(u_i)$。

Step2　使用牛顿插值或拉格朗日插值计算由点$[(u_1,v_1),(u_2,v_2),\cdots,(u_d,v_d)]$确定的插值多项式 P_j。

3.3　判定多元多项式项数的行列式检测法

当多元多项式 P 的项数未知时，根据 P 在特定点处的赋值构造 Hankel 矩阵，其行列式可用于判定 P 的准确项数。

算法 3　多项式项数判定法(TD)

输入：插值点的最大下标 i，黑盒多项式 P 在点$(2^j,3^j,\cdots,p_n^j)$处的赋值 $v_j=P(2^j,3^j,\cdots,p_n^j)$，j=0，1，…，i

输出：多项式项数 k

Step 1　令多项式 P 项数 k 的初值为 0。

Step 2　如果 i≥2 且 i%2=0，构造 Hankel 矩阵：

$$V_i=\begin{bmatrix} v_0 & v_1 & \cdots \\ v_1 & v_2 & \cdots \\ \vdots & \vdots & \cdots \\ v_{i/2} & v_{i/2+1} & \cdots \end{bmatrix}$$

Step 3　计算行列式$|V_i|$和$|V_{i-1}|$，如果$|V_i|=0$而$|V_{i-1}|\neq 0$，那么多项式 P 的项数 k：=i/2。

Step 4　返回 k。

3.4　Ben-Or/Tiwari 算法

Ben-Or/Tiwari 算法包含 3 个要素：Blahut 给出的 BCH 代码编码方法、Grigoiev 提出的不同素数替换变元的技术、Pan 提出的多项式求根快速算法。基于以上 3 点，Ben-Or 和 Tiwari 提出了可一次性恢复整个多元多项式的确定性插值算法。在本文给出的新算法中使用 Ben-Or/Tiwari 算法还原 x_1 的系数多项式。

算法 4　Ben-Or/Tiwari 算法(BT)

输入：黑盒多项式 P 的变元个数 n，项数 k，在 2k 个点$(p_1^0,p_2^0,\cdots,p_n^0),\cdots,(p_1^{2k-1},p_2^{2k-1},\cdots,p_n^{2k-1})$处的赋值 $v_i=P(p_1^i,p_2^i,\cdots,p_n^i)$，i=0，1，…，2k−1

输出：所有出现在 P(**X**)中的单项式及其系数

Step 1　构造并求解方程组 $V_k\lambda=-s$，其中$(V_k)_{ij}=v_{i+j-2}$，$(\lambda)_i=\lambda_{i-1}$，$(s)_i=v_{i+k-1}$。

Step 2　构造单变元多项式 $\Lambda(Z)=Z^k+\lambda_{k-1}Z^{k-1}+\cdots+\lambda_1Z^1+\lambda_0$，并求出 $\Lambda(Z)$的根 $m_1,m_2,\cdots,m_k$

Step 3　分解 $m_i=2^{\alpha_{i1}}3^{\alpha_{i2}}p_n^{\alpha_{in}}$。

Step 4　构造并求解线性方程组 MC=v，其中$(M)_{ij}=(m_j)^{i-1}$，$(C)_i=C_i$，$(v)_i=v_{i-1}$。

Step 5　输出多项式 $P=\sum_{i=1}^{k} C_i x_1^{\alpha_{i1}}x_2^{\alpha_{i2}}\cdots x_n^{\alpha_{n1}}$。

对 BT 算法进行时间复杂度分析：求解 Step 1 中的方程组需要 $O(k^3)$；Step 2 中，求 $\Lambda(Z)$的一个根至多需要 $2^{O(dn\log n)}$；找到全部根需要 $O(k^3dn\log n)$；Step 4 的方程组求解也需 $O(k^3)$。因此，BT 算法的时间复杂度为 $O(k^3dn\log n)$。

3.5　稀疏多元多项式插值分治算法

基于 4 个子算法，给出稀疏多元多项式插值的分治算法，如算法 5 所示。

算法 5　稀疏多元多项式插值问题的分治算法(MIDC)

输入：包含 n 个变元的稀疏多元多项式黑盒 $P(\mathbf{X})=\sum_{i=1}^{t} C_iM_i(x_1,x_2,\cdots,x_n)=\sum_{i=1}^{s} f_i(x_2,x_3,\cdots,x_n)x_1^{e_i}$

输出：所有出现在 P(**X**)中的单项式及其系数

Step 1　使用算法 1 构造关于变元 x_1 在点(1，1，…，1)处的单项式：$P_0=\sum_{j=1}^{s} f_j(1,1,\cdots,1)x_1^{e_j}$。令 $d=\max\{e_j\}$，$v_{0j}=f_j(1,1,\cdots,1)$，j=1，2，…，s。

Step 2　令 i 表示插值点幂次，令 c_j 表示是否完成 f_j 的插值，t_j 表示 f_j 的项数。初值设置为 i=0，c_j=0，t_j=0，j=1，2，…，s。

Step 3　如果对任意 j，c_j=1，转到 Step 4；否则，i=i+1，使用算法 2 获得 $P_i=\sum_{j=1}^{s} f_j(p_1^i,p_2^i,\cdots,p_{n-1}^i)x_1^{e_j}$。令 $v_{ij}=f_j(p_1^i,p_2^i,\cdots,p_{n-1}^i)$。

如果 i%2=0，那么

如果 $c_j=0$ 且 $TD(i,v_{kj})\neq 0$，那么

f_j：=$BT(n-1,TD(i,v_{kj}),v_{kj})$

其中，k=0，1，…，i。转到 Step 3。

Step 4　输出 $P=\sum_{j=1}^{s} f_i(x_2,x_3,\cdots,x_n)x_1^{e_i}$。

注：所有运算均在有限域上进行，可使用有理数恢复算法恢复系数。

算法 ETUP 和 UP 的时间复杂度为 $O(d)$，共执行 $2\tau+1$ 次，其中 $\tau=\max\{\mathrm{term}(f_i)\}$，$i=1,2,\cdots,s$，$\mathrm{term}(f_i)$表示 f_i 的项数。TD 的算法复杂度至多为 $O(\tau)$，至多执行 $O(s\tau)$次；BT 的时间复杂度为 $O(\tau^3dn\log n)$。因此，稀疏多元多项式插值问题分治算法 MIDC 的时间复杂度为：$O(\tau d)+O(s\tau)+O(\tau^3dn\log n)$。

分治后 x_1 的系数多项式的最大项数 τ 远远小于目标多项式的项目 t，x_1 的次数 d 远远小于目标多项式的全次数。通过分治将原问题分解为项数和次数均远小于原多项式的若干个插值问题，相应的代数操作也分解为小规模运算。使用分治策略后插值算法的效率将有较大提升，数值实验见第 4 节。

3.6　实例

令黑盒多项式为：

$$\begin{aligned} P:=&96x_1^2x_2x_3^3-48x_1^3x_2^2+62x_1^3x_2x_3+37x_2^2x_3x_1^2+\\ &5x_2x_3^4-19x_2^2x_3\\ =&(62x_2x_3-48x_2^2)x_1^3+(96x_2x_3^3+37x_2^2x_3)x_1^2+\\ &5x_2x_3^4-19x_2^2x_3 \end{aligned}$$

模 p：=100003

首先由算法 ETUP 首先构造关于变元 x_1 在点(1，1，…，1)处的单项式 $P_0=62x_1^3+133x_1^2+947$；然后使用算法 UP 获得特定赋值点$(2^i,3^i)(i=1,2,\cdots)$处的单变元多项式 $P_i=$

$f_1(2^i,3^i)x_1^3+f_2(2^i,3^i)x_1^2+f_3(2^i,3^i)$。$x_1$ 在多项式 P_i 中的系数如表 2 所列。

表 2 x_1 在多项式 P_i 中的系数

Table 2 Coefficients of in x_1 in P_i

P_i	x_1^3	x_1^2	x_1^0
P_0	14	133	99989
P_1	180	5628	582
P_2	1464	85258	28481
P_3	10320	80027	24172
P_4	68064	36098	40400

利用多项式项数判定法 TD 构造 Hankel 矩阵并计算行列式，例如对于 x_1^3 的各项系数（第 2 列），由 $\mathrm{Det}\left(\begin{bmatrix}14 & 180 & 1464\\ 180 & 1464 & 10320\\ 1464 & 10320 & 68064\end{bmatrix}\right) \bmod 100003=0$，即可判定 x_1^3 的系数多项式为 2 项；接下来用 BT 算法获得 x_1^3 的系数多项式 $62x_2x_3-48x_2^2$。同理可得 x_1^2 和 x_1^0 的系数多项式，最终恢复目标多项式：$P=(62x_2x_3-48x_2^2)x_1^3+(96x_2x_3^3+378x_2^2x_3)x_1^2+5x_2x_3^4-19x_2^2x_3$。

4 数值实验

本节对稀疏多元多项式插值问题分治算法 MIDC、Zippel 算法、Javadi/Monagan 算法（JM）、Ben-Or/Tiwari（BT）算法进行了比较。其中，Zippel 算法是主流计算机代数系统中计算整系数多元多项式 GCD 的主要方法，如 Mathematica，Maple 和 Magma。Ben-Or/Tiwari 算法是一种确定性的可同时对多个变元插值的方法。Javadi/Monagan 算法是 Ben-Or/Tiwari 算法的一种变体，更易于并行实现。

多项式的规模指标包括变元个数、全次数、项数，通过考查不同规模的多项式所需的插值点个数及运行时间可以确定稀疏插值算法的性能。在基本指标确定的情况下，可采用随机多项式对插值算法进行性能比较。算法的正确性仅需要通过插值结果和目标多项式是否一致进行检验，在采用符号计算系统 Maple 环境及有限域下，各个算法均能获得准确结果，无浮点误差。

实验中给出了两组问题集，第 1 组固定变元个数 n 和项数 t，令全次数 d 逐渐增加；第 2 组固定变元个数 n 和全次数 d，令项数 t 逐渐增加。比较的对象为 4 个算法使用的插值点个数及 CPU 时间（单位为 s），编程环境为 MAC 操作系统和 Maple17，硬件环境为 64 位 2.3 GHz Intel core i5 处理器，8 GB 内存。

在测试用例中，黑盒多元多项式的系数属于 $\mathbb{Z}_p$，其中 $p=\mathrm{nextprime}(10^{10})$ 是大于 10^{10} 的最小素数。在所有测试用例中，黑盒仅用于计算输出给定点处的函数值，Zippel 算法和 Javadi/Monagan 算法取定次数界为黑盒多项式全次数 d，Ben-Or/Tiwari 算法取定项数界为黑盒多项式项数 t，分治算法 MIDC 无须给定任何上界信息。

4.1 测试集 1

本组问题集包含 5 个变元的 5 个多元多项式，项数 $t=16$ 次数从 10 变化到 50。第 $i(1\leqslant i\leqslant 5)$ 个多项式使用 Maple 命令 randpoly 随机生成，如测试集 1 第 1 个多项式的生成命令为："> randpoly([x_1, x_2, x_3, x_4, x_5], terms = 16, degree = 10) mod p;"。

表 3 和表 4 分别列出了测试集 1 中 4 个算法 MIDC，BT，JM，Zippel 的运行时间（单位为 s）和使用的插值点个数。

表 3 测试集 1 上的 CPU 时间（$n=5,t=16$）

Table 3 CPU time on Benchmark 1（$n=5,t=16$）

d	MIDC	BT	JM	Zippel
10	0.063	0.132	0.507	0.180
20	0.082	0.141	0.724	0.309
30	0.115	0.146	0.938	0.432
40	0.145	0.208	1.337	0.692
50	0.205	0.232	1.794	0.709

从表 3 可以看出，MIDC 算法和 BT 算法的耗时较少，对次数变化不敏感；而 JM 算法和 Zippel 算法的耗时较多，并且随着次数的增加时间增长较快。因为 JM 算法判断变元次数需要以 d 为界，而 Zippel 算法每恢复一个变元，需要建立的方程组个数为 d，所以次数变化对 JM 和 Zippel 算法的效率有显著影响。

表 4 测试集 1 上的插值点个数（$n=5,t=16$）

Table 4 Number of probeson Benchmark 1（$n=5,t=16$）

d	MIDC	BT	JM	Zippel
10	79	34	192	594
20	109	34	192	1155
30	120	34	192	1736
40	197	34	192	2255
50	260	34	192	2703

从表 4 可以看出，在变元个数和项数固定的情形下，随着次数的增加，MIDC 算法中的插值点也逐渐增加。由于赋值操作的耗时远远少于其他代数操作（如方程组求解）的耗时，因此 MIDC 算法的时间复杂度受其影响极小。BT 算法和 JM 算法的插值点个数是一个关于 t 的函数，因而插值点个数是固定的。

4.2 测试集 2

本组问题集包含 5 个变元的 5 个多元多项式，全次 $d=30$，项数从 4 变化到 64。第 $i(1\leqslant i\leqslant 5)$ 个多项式使用 Maple 命令 randpoly 随机生成，如测试集 2 第 1 个多项式的生成命令为："> randpoly([x_1, x_2, x_3, x_4, $x5$], terms = 4, degree = 30) mod p;"。

表 5 和表 6 分别列出了测试集 2 中 4 个算法 MIDC，BT，JM，Zippel 的运行时间（单位为 s）和使用的插值点个数。

表 5 测试集 2 上的 CPU 时间（$n=5,d=30$）

Table 5 CPU time on Benchmark 2（$n=5,d=30$）

t	MIDC	BT	JM	Zippel
4	0.038	0.038	0.109	0.056
8	0.043	0.052	0.267	0.091
16	0.120	0.184	1.137	0.468
32	0.272	0.981	5.032	3.178
64	0.351	12.250	24.055	35.136

从表 5 可以看出，在多项式规模逐渐增大时，MIDC 算法的时间性能与其他 3 个算法相比具有明显优势。算法 BT，JM，Zippel 的时间复杂度均与项数 t 相关，在项数较多时时间

明显增加。尽管 MIDC 使用 BT 为子算法，但是由于分治策略导致每个系数多项式的项数比原多项式少得多，因此其时间复杂度仍然可以控制在较低的程度。

表 6　测试集 2 上的插值点个数($n=5,d=30$)

Table 6　Number of probes on Benchmark 2 ($n=5,d=30$)

t	MIDC	BT	JM	Zippel
4	31	10	48	527
8	55	18	96	868
16	100	34	192	1860
32	235	66	384	3286
64	541	130	768	6324

MIDC 算法的插值点个数与次数、项数均有关系，插值点个数比 BT 多，而比 JM 和 Zippel 算法都要少。

数值实验表明，分治策略能有效地将原问题分解为若干个规模远小于原问题的子问题，因而在规模越大的插值问题上时间性能的提升越明显；同时，采用 BT 算法插值子多项式也对插值点个数进行了有效的控制。

结束语　本文给出了有限域上求解稀疏多元多项式插值的分治算法，将原问题分解为若干单变元多项式和规模较小的多元多项式的插值，最后合并系数多项式得到原问题的解。本文设计了 4 个子算法，分别是未知次数的基于提前终止策略的单变元多项式插值、已知次数的单变元多项式插值、多元多项式项数判定的 Hankel 矩阵行列式测试法、已知项数的 Ben-Or/Tiwari 算法。算法 1 消除了必须给定次数界的限制。算法 3 消除了必须给定项数界的限制，仅根据黑盒赋值恢复目标多项式。算法 2 和算法 3 实现了单变元和多元多项式插值。在有限域上进行各种代数运算(多项式赋值、线性方程组求解、单变元多项式整数根计算等)有助于提高运算速度，最后使用有理数恢复算法即可复原目标多项式。数值实验中，通过次数和项数的变化测试本文算法、Javadi/Monagan 算法、Ben-Or/Tiwari 算法、Zippel 算法的时间性能和使用的插值点个数。结果表明：本文算法在插值点个数适中的情况下，能在较短时间内求解较大规模的黑盒多元多项式插值问题，具有内在的可并行性；算法 2 可对单变元多项式同时插值，算法 4 可对系数多项式同时插值，体现出分治策略为算法的可并行性提供了有利条件，这也是进一步提高算法效率的有效途径。

参 考 文 献

[1] CUYT A, LEE W. Sparse interpolation and rational approximation[M]// Modern Trends in Constructive Function Theory. 2016: 229-242.

[2] CUYT A, LEE W, WANG X. On tensor decomposition, sparse interpolation and Padé approximation[J]. Jaen Journal on Approximation, 2016, 8(1): 33-58.

[3] HU J, MONAGAN M. A fast parallel sparse polynomial GCD algorithm[J]. Journal of the ACM, 2017, 1(1): 1-41.

[4] MONAGAN M, TUNCER B. Using Sparse Interpolation in Hensel Lifting[C]// International Workshop on Computer Algebra in Scientific Computing. Bucharest: Springer, 2016: 381-400.

[5] BRIANI M, CUYT A, LEE W. Sparse Interpolation, the FFT Algorithm and FIR Filters[C]// International Workshop on Computer Algebra in Scientific Computing. Cham: Springer, 2017: 27-39.

[6] PLONKA G, WANNENWETSCH K, CUYT A, et al. Deterministic sparse FFT for M-sparse vectors[J]. Numerical Algorithms, 2018, 78(1): 133-159.

[7] ISTRATOV A, VYVENKO O. Exponential analysis in physical phenomena[J]. Review of Scientific Instruments, 1999, 70(2): 1233-1257.

[8] ZIPPEL R. Probabilistic algorithms for sparse polynomials. Symbolic and Algebraic Computation[M]// Symbolic and Algebraic Computation. Berlin: Springer, 1979: 216-226.

[9] BEN-OR M, TIWARI P. A deterministic algorithm for sparse multivariate polynomial interpolation[C]// Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing. New York: ACM, 1988: 301-309.

[10] GRIGORIEV D, KARPINSKI M, SINGER M. Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields[J]. SIAM Journal on Computing, 1990, 19(6): 1059-1063.

[11] HUANG M, RAO A. Interpolation of sparse multivariate polynomials over large finite fields with applications[J]. Journal of Algorithms, 1999, 33(2): 204-228.

[12] RAYES M, WANG P, WEBER K. Parallelization of the sparse modular GCD algorithm for multivariate polynomials on shared memory multiprocessors[C]// Proceedings of the International Symposium on Symbolic and Algebraic Computation. New York, NY, USA: ACM, 1994: 66-73.

[13] KALTOFEN E, LEE W. Early termination in sparse interpolation algorithms[J]. Journal of Symbolic Computation, 2003, 36(3-4): 365-400.

[14] KALTOFEN E, LEE W, LOBO A. Early termination in Ben-or/Tiwari sparse interpolation and a hybrid of Zippel's algorithm[C]// Proceedings of the International Symposium on Symbolic and Algebraic Computation. New York, ACM: 2000: 192-201.

[15] JAVADI S, MONAGAN M. Parallel sparse polynomial interpolation over finite fields[C]// Proceedings of the 4th International Workshop on Parallel and Symbolic Computation. New York, ACM: 2010: 160-168.

[16] ARNOLD A, GIESBRECHT M, ROCHE D. Faster sparse multivariate polynomial interpolation of straight-line programs[J]. Journal of Symbolic Computation, 2016, 75: 4-24.

[17] ARNOLD A, GIESBRECHT M, ROCHE D. Sparse interpolation over finite fields via low-order roots of unity[C]// International Symposium on Symbolic and Algebraic Computation. ACM, 2014: 27-34.

[18] CUYT A, LEE W. Multivariate exponential analysis from the minimal number of samples[J]. Advances in Computational Mathematics, 2017(9): 1-16.

[19] HAO Z, KALTOFEN E, ZHI L. Numerical Sparsity Determination and Early Termination[C]// ACM on International Symposium on Symbolic and Algebraic Computation. ACM, 2016: 247-254.

[20] HUANG Q L. An improved early termination sparse interpolation algorithm for multivariate polynomials[J]. Journal of Systems Science and Complexity, 2018, 31(2): 1-13.