有限体上の多変数多項式の因数分解について

野呂 正行 (神戸大・理)
横山 和弘 (九州大・数理)

1　有限体上の多変数多項式の因数分解

正標数での準素イデアル分解を実装しようとすると, 有限体上の多変数多項式の因数分解が必要となる. また, Reed-Solomon 符号の復号法の一つである list decoding においては, 有限体上の二変数多項式の因数分解が用いられる. あるいは, 標数 0 の問題においても, いわゆる楽屋裏の計算として, 有限体上の多変数多項式の因数分解が必要となることはしばしばある. しかし, 残念ながら, Risa/Asir においては, 有限体上の因数分解としては, 一変数の場合の実装しかなかった. これは, 通常の無平方分解アルゴリズムがそのままでは適用できないことと, 体の位数が小さい場合特有の困難があるためであった. しかし, 前者に関しては, Bernardin [3] により, 有限体上での無平方分解アルゴリズムが提案された. 後者の困難とは, 無平方分解, 既約因数分解に必要となる evaluation point が不足することだが, これも, 本稿で提案するように, 体を原始根で表現し, 必要なら代数拡大することにより, 効率を落さずに evaluation point を増やすことができる. これらにより, Risa/Asir 上に, 有限体上の二変数多項式の因数分解の効率よい実装を行うことができた.

現時点では, 位数の小さい有限体上における多変数の無平方分解および二変数の因数分解の実装が完成している. さらに, 因数分解の困難なタイプの多項式に適用できる, あたらしい多項式時間アルゴリズムも得ており, その実装実験結果についても述べる.

2　多変数多項式の無平方分解

現在 Risa/Asir で用いているアルゴリズムは, 以下に述べるように, Bernardin の無平方分解アルゴリズムを modify したものである.

F を標数 p の有限体とし, $f \in F[x_1, \ldots, x_n]$ とする. $'$ が d/dx_1 を表すとする.

$$f = FGH, F = \prod f_i^{a_i}, G = \prod g_j^{b_j}, H = \prod h_k^{c_k}$$

(f_i, g_j, h_k は無平方, 互いに素で, $f_i' \neq 0$, $p \not| a_j$, $p | b_j$, $h_k' = 0$) と書くと $f' = F'GH$ が成り立つ. すると

$$GCD(f, f') = GCD(F, F')GH$$

で, $GCD(F, F') = \prod f_i^{a_i - 1}$ だから

$$f/GCD(f, f') = \prod f_i.$$

$\prod f_i$ で f を繰り返し割り, 割り切れなくなった段階で, その商と $\prod f_i$ の GCD を計算することで, F 中の重複度最小の因子 f_1 が求まる. これを繰り返すと F が全て無平方分解できる. 残りの f は $f = GH$ と書ける. ここで $f' = 0$ が成り立つことに注意する. 以上の手続きを各 x_i について繰り返して残った f は,

$$df/dx_1 = \ldots = df/dx_n = 0$$

を満たす．これは, 全ての指数が p で割り切れることを意味する．すると, F は標数 p の有限体だから, $f = g^p$ と書けることになる．この g に対して, 以上の手続きを再帰的に適用することで, f の無平方分解が得られる．

Bernardin のオリジナルアルゴリズムは, 標数 0 における Yun の方法, すなわち重複度を効率的に計算するアルゴリズムを正標数に応用したもので, 重複度が高い場合に有利であるが, 重複度が p より大きい場合に複雑化するという欠点がある．ここでは, 単純に除算の繰り返しで重複度を求めることとした．我々は, 位数が小さい, 従って, 標数も必然的に小さい場合に効率よく動くアルゴリズムが必要だったため, このような形のアルゴリズムを用いた．

無平方分解のボトルネックは $GCD(f, f')$ の計算である．これを Brown のアルゴリズムで計算しているが, F の位数が小さい場合には evaluation point の不足を招く．この場合には, F の拡大体を使うことになる．

3　二変数への帰着

整数係数多変数多項式の因数分解の場合, EZ 法では, n の内 $n-1$ 個の変数に値を代入し, 一変数で因子のタネを作って, n 変数に Hensel 構成する．ここで, 一変数に落しても, ニセ因子はほとんどでないことを期待している．しかし, 有限体の場合, 一変数に落すと有限体上での一変数多項式の因数分解となり, 一般にニセ因子が大量に発生する．この困難は, $Z[x]$ に相当するのは $F[y][x]$ であると考えることにより克服できる．すなわち, $n-2$ 変数に値を代入して, 二変数多項式の因数分解によりタネを作り Hensel 構成を行うのである．これは, Z 上での多変数の Hensel 構成とほぼ同様に行うことができる．

4　有限体の代数拡大の効率的表現

これまで述べたように, evaluation point の確保のため, 有限体の代数拡大が必要となる場合がある．例えば, $F = GF(q)$ の m 次拡大 F_m を F 上既約な m 次多項式 $h(x) \in F[x]$ により $F_m = F[x]/(h(x))$ で表現すると, F_m の元に対する基本演算に F の基本演算を多数行わなければならず, 計算が大変になる．ここで, F_m の位数ががそれなりに大きければ evaluation point は確保できる．このような場合には, F_m を原始根表現すれば, F_m での演算も効率がよいものとなる．すなわち,

$$F_m^{\times} = \{\alpha^i | (0 \le i \le q^m - 2)\}$$

となるような α が存在するので, $F_m^{\times}$ の元は, 指数 i により表現できる．この場合, 乗除算, 冪乗は容易に計算できる．たとえば乗算は,

$$\alpha^i \cdot \alpha^j = \alpha^{i+j \bmod q^m - 1}$$

とすればよい．加減算は, F_m の位数が小さいことを利用して, テーブル参照によって計算する．すなわち, $\alpha^i + 1 = \alpha^{a_i}$ なる a_i を計算し, (i, a_i) をテーブルで保持しておき,

$$\alpha^i + \alpha^j = \alpha^j(\alpha^{i-j} + 1)$$

として計算すればよい．これまでの実験により, F_m のサイズが 2^{16} 程度までなら実用的といえる．すなわち, この大きさまで体を拡大しても, 計算速度はほとんど変わらない．これにより, 計算効率を落さずに, evaluation point を確保することができる．

5 二変数の Hensel 構成および真の因子の探索

二変数多項式の因数分解は, 一方の変数に値を代入して得た一変数多項式を因数分解した因子をタネに, Hensel 構成を行う. 現在の実装では, ある因子とその cofactor の組に対して Hensel 構成を行っている. 簡単のため, $y = 0$ での evaluation 結果から Hensel 構成をする場合を考える.

$$f(x, y) = \prod_{i=1}^{l} f_i(x) \bmod y$$

と分解したとすると,

$$f(x, y) = \prod_{i=1}^{l} f_{k,i}(x) \bmod y^k$$

への Hensel 構成は

$$f(x, y) = f_{k,1} \cdot F_1 \bmod y^k \Rightarrow F_1(x, y) = f_{k,2} \cdot F_2 \bmod y^k \Rightarrow F_2(x, y) = f_{k,3} \cdot F_3 \bmod y^k \cdots$$

と計算していく. これは, 全ての因子を一度に Hensel 構成する場合と比較して単純かつ十分に高速である.

Hensel 構成の後, 真の因子の探索に入るが, degree bound より少し多めに Hensel 構成することで, いわゆる $d-1$ test [1] がよく効く. しかし, 組み合わせ爆発は克服できない. これは, あとで触れる, 多項式時間アルゴリズムによれば, 効率よく因数分解できる場合がある.

6 全次数最小の元の探索による多項式時間因数分解

ここでは, 一変数でのニセ因子が多い場合に効率よく真の因子を見つけることができる多項式時間因数分解法について簡単に述べる. 詳しくは [5] を参照してほしい.

$f(x, y)$ は x について monic とする.

$$f(x, y) \simeq g_k(x, y) h_k(x, y) \bmod y^k$$

に対し, イデアル $I =< g_k, y^k >$ を考えると $\{g_k, y^k\}$ は $y <_{lex} x$ なる lex order でのグレブナー基底となっている. ここで, $g_k | g \bmod y^k$, $g | f$ なる既約因子 g は $g \in I$ をみたす.

定理 1
k が十分大きいとき, g は I の *degree compatible order* $<$ でのグレブナー基底の順序最小の元である.

証明 もし $g' <_{drl} g$, $g' \in I$ があれば $Id(g', g)$ は 0 次元で, Bézout の定理により

$$\#V(Id(g', g)) \leq tdeg(g') \cdot tdeg(g)$$

だが,

$$\#V(I) = k \deg_x(g_k) \leq \#V(Id(g', g))$$

より k を大きくとれば矛盾. ∎

定理により, 十分大きな k に対し, I の元の中で, 全次数の小さいものを見つけることができればそれが f の既約因子となる. ここで, $\{g_k, y^k\}$ が $<_{lex}$ に関するグレブナー基底だから, I へのメンバーシップは未定係数法によりきめることができる. 問題となるのは k の大きさである. 確定的なアルゴリズムとするには k として $k > tdeg(f)^2 / \deg_x(g_k)$ を満たす値を選ぶ必要があるが, 小さい次数の因子の存在が期待できる場合には, k を小さくとって, heuristic なアルゴリズムとすることもできる.

7 元の体での因子の計算

evaluation point を増やすために, 体を $F = GF(q)$ から F_m に拡大した場合, F 上の既約因子が F_m 上で分解する可能性がある. $f \in F[x_1, \ldots, x_n]$, f は F 上既約で $f = \prod f_i$, f_i は F_m 上既約, と分解したとする. F_m/F は Galois 拡大で,

$$G = Gal(F_m/F) = \langle \sigma \rangle,$$

ただし $\sigma : \beta \mapsto \beta^q$ である. S を f_1 の G-orbit とすると $\prod_{s \in S} s$ は G-不変だから

$$\prod_{s \in S} s \in F[x_1, \ldots, x_n].$$

f は F 上既約だから $f = \prod_{s \in S} s$. よって, F 上の既約因子は F_m 上の既約因子の G-orbit に属する元の積である. $\sigma(h)$ の計算は係数を q 乗すればよいから容易である. このようにして, F_m 上での因子から, F 上の因子を得ることができる.

8 タイミングデータ

まず, [4] で扱われている種々の二変数多項式の因数分解について, タイミングデータを示す.

	f_2	f_7	f_{11}	f_{13}	f_{17}	$f_{17}(x, y^2)$
$\deg_x, \deg_y$	7,5	100,100	300,300	$10^3, 10^3$	32,16	32,32
#factors	2	2	1	2	1	2
Maple7	3.7	26	25	10420	33	30
Asir	0 (2^3)	2.0	34	5040	0.24	0.57

表で f_p は $GF(p)$ 上の二変数多項式である. Maple7, Asir による PentiumIII 1GHz 上での計算時間を秒で表示してある. Asir の (p^m) は $GF(p^m)$ まで拡大する必要があったことを示す. [2] によれば, 素体上の多項式演算は, Maple kernel で特別な実装が行われており, Maple と Asir の計算時間を比較することは unfair ではないと考える. Maple との比較でみると, 次のようなことがわかる.

- Hensel 構成の性能比較

  f_{11}, f_{13} は Hensel 構成がほとんどを占めている. このことから Hensel 構成自体の性能は大差ないとみてよいであろう.

- 解探索部の性能比較

  f_{17}, $f_{17}(x, y^2)$ については, 計算時間のほとんどは bad combination の排除である. これは $d-1$ test がうまくいっている例と考えられる.

- 体の拡大が必要な場合の比較

  Maple では, 体の拡大が必要な場合には, Domains package (一般の有限体を扱う generic な実装) を使うため, f_2, f_7 で差が出たと考えられる.

つぎに, 体の拡大と効率の関係について示す. 表は, f_7 および, $f_{17}(x, y^2)$ について, 拡大次数を上げていった場合の計算時間を示す.

拡大次数	1	2	3	4	5	6	7
f_7	—	2.6	2.9	2.7	2.6	5.8	9.1
$f_{17}(x, y^2)$	0.57	0.78	0.55	2.3	—	—	—

表より次のようなことが観察される.

- 体が小さいうちは, 拡大しても効率は落ちない

  もちろん, 拡大により因子が増える場合は除く.

- 体のサイズが大きくなると, 急激に効率が落ちる

  参照するテーブルが大きくなるせいと考えられる.

最後に, 一変数でのニセ因子が多く, 真の因子の探索において組み合わせ爆発を起こす場合について, 今回提案した多項式時間アルゴリズムの効果を示す.

$$f(x, y) = f_{17}(x, y^2) f_{17}(x + 1, y^2)$$

は, 真の因子は 4 個だが, mod$y$ での因子は 32 個である. これを, 通常の Belrekamp-Zassenhaus 型で計算した場合には,

- 処理した組み合わせ：6626138
- $d - 1$ test で排除：6234991
- 計算時間：558 秒

一方, 新アルゴリズムで計算した場合, 次数の上限を 16 とした場合 2.8 秒, 32 とした場合 46 秒であった. 前者は甘すぎる bound だが, 後者は少なくとも因子は得られる bound であり, 新アルゴリズムが実用的であることがわかる.

9 今後の予定

現在, 多変数の因数分解を実装中であり, 完成後は正標数の準素イデアル分解の実装を行う予定である.

参 考 文 献

[1] Abbott, J., Shoup, V., Zimmermann, P. (2000). Factorization in $\mathbb{Z}[x]$: the searching phase. *Proc. ISSAC2000*, 1–7.

[2] Bernardin, L., Monagan, M.B. (1997). Efficient multivariate factorization over finite fields. *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes (AAECC-12). Lecture Notes in Computer Science* **1255**, 15–28.

[3] Bernardin, L. (1997). Factorization Examples. `http://www.inf.ethz.ch/personal/bernardi/factor/`.

[4] Bernardin, L. (1997). On square-free factorization of multivariate polynomials over a finite field. *Theoret. Comput. Sci.* **187**, 105–116.

[5] Noro, M. and Yokoyama, K. (2002). Yet Another Practical Implementation of Polynomial Factorization over Finite Fields. *Proc. ISSAC2002*, 2002–206.