数理解析研究所講究録
第1652巻 2009年 19-25



# グレブナー walk アルゴリズムの実装と効率化について

野呂 正行
MASAYUKI NORO
神戸大学大学院理学研究科*

崎山 裕尊
HIROTAKA SAKIYAMA
神戸大学大学院自然科学研究科†

**Abstract**

グレブナー walk アルゴリズムは, ある項順序でのグレブナー基底から他のグレブナー項順序のグレブナー基底を計算する change of orderging アルゴリズムの一つである. 1997 年に Collart ら [2] が提案して以来, さまざまな改良が行われてきた ([3][4][5]). ここでは, グレブナー walk について簡単に紹介し, Risa/Asir での実装, 効率化のための改良およびその効果について述べる.

## 1 Change of ordering アルゴリズム

多項式イデアル $I$ の有限部分集合 $G$ が, ある項順序 $<_1$ に関するグレブナー基底であることがあらかじめ分かっている場合に, 別の項順序 $<_2$ に関するグレブナー基底を計算することを一般に change of ordering と呼ぶ. この方法としては以下のような方法が知られている.

- Buchberger アルゴリズム

  $G$ がグレブナー基底であることを忘れて単に Buchberger アルゴリズムを実行するだけのことだが, 既に $G$ が「良い」性質を持っている場合には計算が効率よくできる場合がある.

- FGLM アルゴリズム

  $I$ が 0 次元の場合には, 線形代数を用いて $<_2$ のグレブナー基底を計算することができる.

- modular change of ordering

  係数体が $\mathbf{Q}$ の場合, $\mathrm{in}_{<_1}(G)$ を割らない素数 $p$ をとり, $\mathbf{F}_p$ 上で $\langle G \rangle$ の $<_2$ に関するグレブナー基底を計算して, その要素と同じサポートを持つ $I$ の元を求めることで, $<_2$ に関するグレブナー基底を求めることができる. 求め方としては, Buchberger trace アルゴリズムで候補生成のみを行う, あるいは未定係数法により線形代数に持ち込む方法などがある.

- Hilbert driven アルゴリズム

  $I$ が斉次の場合, $K[x]/I$ が次数環になり, その各斉次成分の次元が Hilbert 関数により分かることを用いて, Buchberger アルゴリズム実行における userless pair の計算を省く方法である.

- グレブナー walk アルゴリズム

  $<_1$, $<_2$ と compatible な weight vector (後述) を選び, それらを結ぶ線分と交わる グレブナー cone (後述) に対応するグレブナー基底を次々と計算していく方法である. weight vector がうまくとってあれば, 手間の少ない Buchberger アルゴリズム実行と, inter-reduction が主たる計算となる.

*noro@math.kobe-u.ac.jp
†sakiyama@math.kobe-u.ac.jp

それぞれに特色があり，入力に応じて使い分けられる．これまで Risa/Asir では，modular change of ordering の実装，改良を行ってきたが，特に 0 次元でないイデアルの場合に効率に不満があった．最近，Fukuda らにより新しいグレブナー walk アルゴリズム (generic walk [5]) が考案された．これは，これまで提案された weight の摂動方法の曖昧さを除く方法を提案しており理論的に興味深く，また，実用性も感じられたため，Risa/Asir に実装して実験を行った．その過程で，効率上の問題が発生したが，それに対する一つの改良を提案する．また，常に問題となる，$<_2$ と compatible な weight をどうとるか，について，安直ではあるが有効な方法を提案する．最後に，これらの改良を採り入れた generic walk, walk アルゴリズムを実装した場合の実験結果を示す．

## 2 グレブナー walk アルゴリズム

以下では，多項式 $f$ に対し，項順序 $<$ に関する先頭項を $\mathrm{in}_<(f)$, weight $w \in \mathbf{R}^n_{\geq 0}$ に関する最高次部分を $\mathrm{in}_w(f)$ で表す．また,$K$ を体, $R = K[x_1, \ldots, x_n]$ とし，イデアル $I \subset R$ を固定する．項順序 $<$ に関する $I$ の簡約グレブナー基底を $\mathrm{GB}_<(I)$ と書く．$\langle \mathrm{in}_w(f) \mid f \in I \rangle$ を $in_w(I)$ と書く．一般に，多項式集合 $F$, $w \in \mathbf{R}^n_{\geq 0}$ に対し，$\mathrm{in}_<(f) = \mathrm{in}_<(\mathrm{in}_w(f))$ が全ての $f \in F$ について成り立つとき，$w$ は $<$ について $F$ と compatible であるという．$C_<(I) = \{w \in \mathbf{R}^n_{\geq 0} \mid w$ は $<$ について $\mathrm{GB}_<(I)$ と compatible$\}$ と定義し，$<$ 上のグレブナー cone と呼ぶ．

$$C_<(I) = \{w \in \mathbf{R}^n_{\geq 0} \mid f = cx^\alpha + \sum_{\beta < \alpha} c_\beta x^\beta \in \mathrm{GB}_<(I) \text{ に現れる全ての} \beta \text{に対し } \langle w, \alpha \rangle \geq \langle w, \beta \rangle\} \quad (1)$$

だから，$C_<(I)$ は凸多角錐である．あらゆる項順序に関するグレブナー cone およびその face を全部集めた集合を $GF(I)$ と書く．

**定理 1 (Mora-Robbiano [1])**
$GF(I)$ は *fan* である．

特に，$GF(I)$ は有限集合であり，グレブナー cone の集合と，$\{\mathrm{in}_<(I) \mid <$ は項順序 $\}$ の集合は 1 対 1 に対応する．後者は，相異なる $\mathrm{GB}_<(I)$ の集合と考えてもよい．以下で，weight $w$ で比較し，項順序 $<$ を tie breaker とする項順序を $w+<$ と書く．

グレブナー walk アルゴリズムは，項順序 $<_1$, $<_2$ に対し $\mathrm{GB}_{<_1}(I)$ から $\mathrm{GB}_{<_2}(I)$ を計算する change of ordering アルゴリズムである．$w_1 \in C_{<_1}$, $w_2 \in C_{<_2}$ を選び，$w_1$ と $w_2$ を結ぶ線分 $L : w(t) = (1-t)w_1 + tw_2$ $(1 \leq t \leq 1)$ を考える．この線分は，有限個のグレブナー cone と交わる．すなわち，$0 = t_0 < t_1 < \cdots < t_m = 1$ および相異なるグレブナー cone $C_1, \ldots C_m$ が存在して，$L = L_1 \cup L_2 \cup \cdots \cup L_m$, $L_i = \{w(t) \mid t_{i-1} \leq t \leq t_i\} \subset C_i$．ここで，$L_i$ が 2 つ以上の cone に属する場合には，$w(t_i)+ <_2$ に対応する cone を選ぶものとする．次の定理がグレブナー walk の基本である．

**定理 2 (cf. Fukuda et al.[5] Proposition 5)**
$<, <'$ を項順序とし，*weight* $w$ が $w \in C_< \cap C_{<'}$ を満たすとし，$G$ を イデアル $I$ の $<$ に関するグレブナー基底とする．このとき次が成り立つ．

1. $\mathrm{GB}_<(\mathrm{in}_w(I)) = \{\mathrm{in}_w(g) \mid g \in G\}$
2. $H = \mathrm{GB}_{<'}(\mathrm{in}_w(I))$ ならば，$\{f - \mathbf{NF}_G(f) \mid f \in H\}$ は $I$ の $w+<'$ に関する極小グレブナー基底．
3. $\mathrm{GB}_{<'}(I) = \mathrm{GB}_{w+<'}(I)$.

この定理により, 隣り合う cone に対応するグレブナー基底を, $\mathrm{GB}_{<}(\mathrm{in}_w(I))$ を仲立ちとして変換することができる. $<'$ に関する極小グレブナー基底を interreduce (相互簡約) して簡約グレブナー基底を得る操作を $convert(G, w, <, <')$ と書く.

**アルゴリズム 1 (グレブナー walk アルゴリズム)**

*if* $\mathrm{in}_{<_1}(G_0) \neq \mathrm{in}_{<_2}(\mathrm{in}_{w_1}(G_0))$ *then*
  $G \leftarrow convert(G_0, w_1, <_1, w_1+ <_2)$
*else*
  $G \leftarrow G_0$
*endif*
$t_0 \leftarrow 0$
*do*
  $t_1 \leftarrow \max\{s > t_0 \mid w(s) \in C_{w(t_0)+<_2}\}$
  $G \leftarrow convert(G, w(t_1), w(t_0)+ <_2, w(t_1)+ <_2)$
  *if* $t_1 \geq 1$ *return* $G$
  $t_0 \leftarrow t_1$
*end do*

## 3 generic グレブナー walk

前節のアルゴリズムにおいては, *convert*() の効率が全体の効率を左右する. *convert*() においては, $\mathrm{in}_w(I)$ のグレブナー基底計算の入力となる $\{\mathrm{in}_w(g) \mid g \in G\}$ に現れる多項式の複雑さが問題となる. 考えるべき $w$ はグレブナー cone の boundary 上にある. この場合, $w$ を含む face の数が多い程, $\mathrm{in}_w(g)$ の項数が増える可能性がある.

**例 1**

$G = \{x + y_1 + \cdots + y_l, f_2(y), \ldots, f_k(y)\}$, $y = (y_1, \ldots, y_l)$ で, $G$ が $x > y_1 > \cdots > y_l$ なる全次数辞書式順序 $<_1$ での簡約グレブナー基底とする. $\langle G \rangle$ の辞書式順序 $<_2$ でのグレブナー基底を計算する場合, $w_1 = (1, 1, \ldots, 1)$, $w_2 = (1, 0, \ldots, 0)$ を選ぶことができる. このとき, 線分 $L$ は $C_{<_1}(I)$ に含まれる. よって, ループ内の最初のステップで $t_1 = 1$ となる. すると, $G$ の $w_2$-斉次な最高次部分を抜き出すことになるが, この場合, $f_2(y), \ldots, f_k(y)$ がそのまま出て来ることになり, 単なる $<_2$ に関するグレブナー基底計算と変わらないことになる.

この欠点を回避するため, 各ステップで $w(t_1)$ を摂動させ, 線分 $L_i$ がグレブナー cone と少ない個数の face と交わるようにする方法が提案された (fractal walk [3]). 原理は次の命題である.

**命題 3**

項順序 $<$ が *weight* の列 $(\omega_1, \ldots, \omega_n)$ による辞書式順序として与えれているとする. このとき

$$\omega_\epsilon = \omega_1 + \epsilon\omega_2 + \ldots + \epsilon^{n-1}\omega_n \tag{2}$$

とおくと, $\epsilon > 0$ を十分小さくとれば, $\omega_\epsilon$ は $<$ と *compatible* で $C_<(I)$ の内部にある.

よって, $L_i$ の始点あるいは終点 (あるいは双方) を十分小さい $\epsilon$ によりこの形に摂動させればそうすれば, 直線 (折れ線) L が cone を出るとき, 1 つの facet のみ通過することが期待できる. ここで, 始点の摂動は, 既

に始点が現在のグレブナー基底と compatible なので計算で可能となるが, 終点の摂動は, 新しいグレブナー基底がないと計算できないので, fractal walk では heuristic な試行錯誤で行われている. 特に $w_2$ がいくつかの face の交わり上にある場合, 最終ステップを効率よく計算するには $w_2$ の摂動が必須である. Tran Q. Nam [4] は, グレブナー基底の元の degree bound を元に, 摂動を deterministic に与えているが, weight の成分が非現実的な大きさになるため, 実用的には問題がある. generic walk [5] では, この問題を, $w_1$, $w_2$ の「無限小」摂動を導入することで解決した. $<_1$, $<_2$ を表す weight の列 $(\omega_1,\ldots,\omega_n)$, $(\tau_1,\ldots,\tau_n)$ から (2) により $\omega_\delta$, $\tau_\epsilon$ を作る. このとき, $\delta$, $\epsilon$ を十分小さくとれば, $\omega_\delta$, $\tau_\epsilon$ がそれぞれ $C_{<_1}$, $C_{<_2}$ の内部にあり, これらを結ぶ線分 $L_{\delta,\epsilon}$ がグレブナー cone の内部を通過しながら, cone を出るときには常にただ一つの facet と交わることが証明できる.

更に, $\delta$, $\epsilon$ を十分小さくとることにより, walk の各ステップにおいて, $L_{\delta,\epsilon}$ が cone を出るときにどの facet と交差するかを, $<_1$, $<_2$ を表す weight を用いた計算のみで求めることができる. 各 facet はある vector $v=\alpha-\beta\in\mathbf{Z}^n$ ($\alpha,\beta$ は (1) に現れる vector) により $F_v=\{w\in\mathbf{R}_{\geq 0}\mid\langle w,v\rangle=0\}$ と表される. $u,v\in\mathbf{Z}^n$ に対し, facet preorder $\prec$ を

$$u\prec v\Leftrightarrow(\langle\tau_1,u\rangle v,\ldots,\langle\tau_n,u\rangle v)<_{1,lex}(\langle\tau_1,v\rangle u,\ldots,\langle\tau_n,v\rangle u)\tag{3}$$

で定義する. ここで, $<_{1,lex}$ は $<_1$ を左の成分から辞書式に適用することを意味する. すると, walk の各ステップにおける cone の facet のうち, 進行方向にあって最初に $L_{\delta,\epsilon}$ が交差する facet は, facet preorder で最小のものであることが証明できる. 注目すべきことは, 実際に交点での weight $w$ を求めることなしに, 交差する facet $F_v$ を求めることができることである. さらに, この $w$ に対し, $\mathrm{in}_w(g)=\{c_\alpha x^\alpha+\sum_\beta c_\beta x^\beta\mid x^\alpha$は$g$の先頭項,$\alpha-\beta\prec v$かつ$v\prec\alpha-\beta\}$ となり $\mathrm{in}_w(g)$ の計算にも $w$ は不要である. *convert* においては, 現在のグレブナー基底 $G$ に関する剰余計算が必要である. 途中の weight がないと項順序が定められないが, $G$ の先頭項は常に知られていることに注意する.

**定義 4**
$G$ をある項順序 $<$ に関する $I$ のグレブナー基底とするとき $\{(g,\mathrm{in}_<(g))\mid g\in G\}$ を marked グレブナー基底と呼ぶ.

marked グレブナー基底が与えられればそれによる剰余計算は, 項順序なしで可能である. このようにして, 形式的に $L_{\delta,\epsilon}$ を考え, それらが交差する cone をたどりながら walk を実行するのが generic walk アルゴリズムである. この場合, $\mathrm{in}_w(g)$ に現れる項 $x^\beta$ は $\alpha-\beta$ が平行であるものに限られる. これはどのような weight を用いても同時に含まれるので, $\mathrm{in}_w(g)$ の項数は最小に押えられている.

## 4 generic グレブナー walk の実装と問題点

Fukuda et al. [5] では, ある種の toric イデアル (lattice イデアル) に対する実装とその効果が述べられている. Risa/Asir 上への実装においては, まず Asir 言語でプロトタイプを書き, weight の計算, marked グレブナー基底による剰余計算など, 効率に影響を与える関数を組み込みとした.

- `dp_compute_last_w(`$G,H,W,M1,M2$`)`

  現在の cone に対応する marked グレブナー基底 $(G,H)$ を与え, $L_{\delta,\epsilon}$ が facet $F_W$ の次に交差する facet を表す vector を求める. $M1$, $M2$ は $<_1$, $<_2$ を表す行列である.

- `dp_true_nf_mared(`$Ind,F,G,H$`)`

  グレブナー基底による剰余を求める `dp_true_nf` の marked グレブナー基底版である.

しかし, 我々の最初の実装で, より一般的なイデアルに対して辞書式順序への変換を試したところ, $\mathrm{GB}_{<_2}(\mathrm{in}_w(I))$ の計算は常に高速であるが, marked グレブナー基底による剰余計算が大変時間がかかる場合がしばしばあることが分かった. これは, dividend に現れる項が正しい項順序で整列されていないため, 低い順序の項を無駄に書き換えてしまうことが起こっているためである. これを解決する方法として, やや本末転倒かつ安直ではあるが, marked グレブナー基底 $G$ から, それを与えるような weight を計算し, その weight により剰余計算を行うことにした. これは, 連立不等式系

$$\{\langle w, \alpha - \beta\rangle > 0 \mid (g, x^\alpha) \in G, x^\beta \in \mathrm{Supp}(g) \setminus \{x^\alpha\}\} \tag{4}$$

の解を求めることを意味する. 経験上, 不等式の数は数千に達する場合も多いため, 定評ある `cddlib` (Fukuda, 2006) に実装されている, 線形不等式系の内点解を求める関数 `dd_FindRelativeInterior` を, OpenXM サーバ `ox_cdd`, `ox_cddgmp` (藤原 [6]) に追加し使用することにした. 前者は浮動小数による計算, 後者は `gmp` による厳密計算を行う. ここで必要な weight はあくまで剰余計算のヒントであるため, 完全に先頭項を与える必要はないと考えたため, 浮動小数版も用いることにした. この計算は, 各ステップでの interreduce の前に行われる. ここで得た新しいグレブナー基底は, 次のステップでの $\mathrm{GB}_{<_2}(\mathrm{in}_w(I))$ に続く剰余計算でも使われるが, この weight はそこでも用いられる.

# 5 通常の walk における modular 計算の応用

前節では, generic walk をより効率化するために, 敢えて途中で weight を逆算する方法を提案した. ここでは, 元の walk に戻り, スタート, ターゲットの weight $w_1$, $w_2$ を摂動する方法を考える. $w_1$ については, 既にグレブナー基底を知っているため, 前節の線形不等式系 (4) を解くことで cone 内部の vector を得る. 問題は $w_2$ である. 有理数体上での $<_2$ に関するグレブナー基底を $G$ とするとき, 有限体 $\mathbf{F}_p$ 上での計算により, $<_2$ に関するグレブナー基底 $G_p$ はなんらかの方法で効率よく計算できる場合がしばしばある. このとき, 有限個の素数 $p$ を除いて, $G_p$ の各元のサポートは, $G$ のそれと一致している. よって, $G_p$ を使った計算で $C_{<_2}$ の内部の vector を推測し, そこをターゲットとして通常の walk を行うという方法が考えられる.

# 6 実験

これまでに述べた方法を実装し, いくつかの例について計算を行った結果を示す. 全て, 全次数逆辞書式順序から辞書式順序への change of ordering を行う.

## 6.1 種々の change of ordering アルゴリズムの比較

- `tolex`

  まず有限体上で, 0 次元イデアルに対しては, FGLM により, 0 次元でない場合には, Buchberger アルゴリズムにより辞書式順序グレブナー基底を計算する. そのサポートをもとに未定係数法と Hensel lifting で有理数体上の基底を計算する.

- `gwalk`

  marked グレブナー基底と compatible な weight を `ox_cdd` により計算し, tie-breaker として全次数逆辞書式順序を用いて interreduce を行う generic グレブナー walk を実行する.

- `mwalk`

  入力のグレブナー基底, および有限体上での辞書式順序グレブナー基底それぞれについて, compatible な weight を `ox_cdd` により計算する. それらに対し, tie-breaker として全次数逆辞書式順序を用いる項順序をそれぞれ設定し, 通常のグレブナー walk を実行する.

計算環境は, Intel Xeon X5365 3GHz, 16GB memory である. 計算時間は実経過時間を秒で表示した. 入力多項式イデアルは [9],[10] から取った. かっこ内の数字は walk のステップ数を表す. `ldim` は, 0 次元イデアルの場合, 剰余環の線形次元を表す. — は, イデアルが 0 次元でないことを表す. これらの例についてい

| | `ldim` | `tolex` | `gwalk` | `mwalk` |
|---|---|---|---|---|
| *assur44* | 56 | 122 | 66(167) | 176(68) |
| *cyclic7* | 924 | 37 | 1590(1520) | 1777(1099) |
| *dl* | 468 | 1142 | 5075(1564) | 8414(963) |
| *eco9* | 128 | 12 | 135(403) | 275(210) |
| *eco10* | 256 | 247 | 5476(996) | 19810(479) |
| *extcyc6* | 936 | 251 | 5556(2289) | 2025(941) |
| *fabrice24* | 40 | 565 | 270(104) | 570(43) |
| *ilias_k_2* | 144 | 86 | 281 (437) | 368(239) |
| *ilias_k_3* | 168 | 427 | 1077 (506) | 2617(318) |
| *jcf26* | 40 | 969 | 563(103) | 1185(43) |
| *katsura7* | 128 | 350 | 856(384) | 2039(203) |
| *kin1-rev* | 48 | 2.5 | 2.1(51) | 2.9(44) |
| *rbpl24* | 40 | 490 | 264(104) | 572(43) |
| *redeco10* | 256 | 249 | 6423(839) | 16260(424) |
| *reimer6* | 576 | 25 | 1942(694) | 215(289) |
| *virasoro* | 256 | 15 | 100 (157) | 46(83) |
| *cohn3* | — | 6708 | 2201(440) | 2136(340) |
| *f855* | — | 17 | 55(83) | 15(61) |
| *hairer2* | — | | | |
| *kotsireas* | — | 10 | 25(228) | 17(146) |
| *rbpl* | — | | | |
| *redcyc7* | — | 99 | 129(370) | 162(274) |

表 1: 種々の change of ordering アルゴリズムの比較

えば, 0 次元イデアルに対しては, 概ね modular FGLM である `tolex` がまだ高速な場合が多いが, いくつかの例では walk が勝っている. 0 次元でない場合の例は少ないが, やはり walk が `tolex` に勝る例がある (*cohn3*). walk の間の比較も, これらの例については `gwalk` がやや有利に見えるが, はっきりした優劣は見られない. 興味深い点は, step 数が必ずしも計算時間と相関しない点である. これは, `gwalk` においてはもともとの weight の近くの facet を通過していくのに比べ, `mwalk` では modular 計算でターゲットの weight を選ぶ場合, この $w_2$ とは相当かけ離れた vector が選ばれている可能性がある. 結果として通過する cone は全く異なっていることもあり得る. よって計算経過は異なり, step 数も異なるのであろう. いずれにせよ, 計算時間の比較をみれば, どのアルゴリズムも捨て難い. CPU 数に余裕があれば, 同時に実行するのが現実的かもしれない. なお, [7] においては, 中間的な項順序への walk (寄り道) を行うことの効果についての実

験も行った. これは, 例えばターゲットが辞書式順序の場合, いくつかの変数を消去する項順序 (ブロック順序) への walk を行い, そこからターゲットへ walk するという方法である. この場合, 中間項順序で消去する変数の個数を heuristic に決める必要があるものの, 直接 walk するより高速になる場合があることが示されている.

## 参 考 文 献


[1] Mora, T., Robbiano, L., The Groebner fan of an ideal. J. Symb. Comp., 6, 183-208 (1988).

[2] Collart, S., Kalkbrener, M., Mall, D., Coverting bases with the Groebner walk. J. Symb. Comp., 24, 443-464 (1997).

[3] Amrhein, B., Gloor, O., The Fractal Walk. Buchberger, B. Winkler, F. (eds), Groebner Bases and Applications, 305-322 (1998).

[4] Tran, Q., A fast alorithm for Groebner basis conversion and its applications. J. Symb. Comp., 30, 451-467 (2000).

[5] Fukuda, K., Jensen, A.N., Lauritzen, N., Thomas, R., The generic Groebner walk. J. Symb. Comp., 42, 298-312 (2007).

[6] 藤原 健志, multivariate resultants の実装と比較. 神戸大学修士論文 (2005).

[7] 崎山 裕尊, 種々の Groebner walk algorithm の実装および改良について. 神戸大学修士論文 (2008).

[8] Fukuda, K., cddlib. http://www.ifor.math.ethz.ch/~fukuda/cdd_home/cdd.html (2007).

[9] http://invo.jinr.ru/.

[10] http://www.symbolicdata.org