The 32nd Annual Conference of the Japanese Society for Artificial Intelligence, 2018

SAT ソルバを参考にした行列のフルランク性の判定手法

Determination method influenced by SAT solver for the full rankness of a matrix

町出 智也 *1*2
Tomoya Machide

薗部 知大 *1*2
Tomohiro Sonobe

*1国立情報学研究所
National Institute of Informatics

*2JST ERATO 河原林巨大グラフプロジェクト
JST, ERATO, Kawarabayashi Large Graph Project, Japan

We introduce a new determination method for the full rankness of a matrix over the binary field, which is made with algorithms used by SAT solver as a reference.

1. はじめに

数を行と列に沿って矩形状に配列したものである行列は、線型代数学において（従ってほとんどの科学的分野において）重要な概念の一つである。ランク（階数）は最も基本的な指標の一つであり、その行列が表す線形方程式の非退化性（解空間の自由度）の尺度として捉えられる。他にも、線形独立な列ベクトルや行ベクトルの最大個数、小行列の最大サイズ、特異値の個数、そして線形写像として考えた時の像空間の次元など、様々な解釈がある。この解釈の多様性は、ランクという指標が重要であり、豊かな性質を備えていることを示している。

SAT（充足可能性問題、satisfiability problem）は判定問題の一種であり、この問題を解くプログラムは SAT ソルバと呼ばれる。SAT は、計算機工学における最も基本的な問題として，システム検証、スケジューリング問題、定理証明など、様々な分野に応用されている。近年、人工知能や機械学習の方法を取り入れることにより、大規模な SAT 問題を非常に高速に解く SAT ソルバが実現され，実用的応用が急速に拡大している [3, 6]。

本論文では、2 元体 $\mathbb{F}_2$ 上の行列のフルランク性を判定する、新たな手法を紹介する。その手法は、ガウスの消去法に SAT ソルバで使われているアルゴリズムを付加することにより得られる。また、オリジナルのガウスの消去法との比較実験も報告する。計算時間が数倍速くなることが確認できる。

なお 2 元体 $\mathbb{F}_2$ とは、2 を法とした余りで演算（加法、減法、乗法、除法）を定めている、要素が 2 つの集合 $\{0, 1\}$ のことである。イメージとしては、要素が 2 個の場合の有理数体 $\mathbb{Q}$ である。2 を任意の素数 p にすることも可能であり、その場合は $\mathbb{F}_p$ と書く。

提案手法は、 ガウスの消去法を土台にしているため、次の拡張が可能である：「(a) 厳密なランク r の値を任意の体上で計算すること」、「(b) r 個の一次独立な行（列）ベクトルを見つけること」。しかし議論を簡明にするため、本論文では「フルランクか否かを 2 元体 $\mathbb{F}_2$ 上で判定する」ことに内容を限定する。拡張に関する詳しい内容は、今後の論文で論じる。

本論文の構成は以下の通りである。第 2. 節では準備として行列を SAT 問題に変換する方法を、第 3. 節では本手法の着想に至った経緯を述べる。第 4. 節では提案手法の詳細を述べ、第 5 節では実験結果を報告する。最後に第 6. 節で 本論文のまとめを述べる。

第 3. 節では、ある行列のフルランク性を SAT 問題に変換して「MiniSat」ソルバ [2] に解かせた場合の出力結果に言及している。その結果は、論文を構成する上では必ずしも必要ではないが、本手法の生みの親であり、本論文で一番重要な内容のように思われる。

連絡先: machide@nii.ac.jp, tomohiro_sonobe@nii.ac.jp.

2. 準備

2 元体 $\mathbb{F}_2$ 上の線形関係式を CNF 形式 (乗法標準形、Conjunctive Normal Form) の命題論理式に変換する方法を述べる。そのため、$\mathbb{F}_2$ の元を

$$1 \leftrightarrow \text{True（真）}, \qquad 0 \leftrightarrow \text{False（偽）}$$

ように本論文では同一視する。なお CNF 形式は、節（clause）と呼ばれる論理和 (“or”, $\vee$) で連結された論理式を、論理積 (“and”, $\wedge$) で結合することで記述される。

任意の命題論理式は必ず CNF 形式で記述できることが知られている。体 $\mathbb{F}_2$ の足し算 $+$ は排他的論理和 “xor” と同じ演算規則を満たすので、$\mathbb{F}_2$ 上線形関係式も CNF 形式の論理式に変換できる。例えば次のように、左の線形関係式を右の CNF 形式に変換できる。

$$\begin{aligned} x = 0 \quad &\Longleftrightarrow \quad \neg x \ = \ \text{True}, \\ x + y = 0 \quad &\Longleftrightarrow \quad (\neg x \vee y) \wedge (x \vee \neg y) \ = \ \text{True}. \end{aligned}$$

記述を簡明にするため、これ以降、命題論理式の “$=$ True” は省略して書くことにする。

一般に、p 変数の線形関係式

$$x_1 + \cdots + x_p = 0 \tag{2.1}$$

は、次の 2^{p-1} 個の節

$$y_1 \vee \cdots \vee y_p \quad \left(\begin{array}{c} y_i \in \{x_i, \neg x_i\}, \\ y_i = \neg x_i \text{ となる個数は奇数個} \end{array} \right) \tag{2.2}$$

の積に変換される。それは以下の理由による：

「線形関係式 (2.1) が成り立つ」
$\Leftrightarrow$「$x_i = 1$ となる個数は偶数である」
$\Leftrightarrow$「$x_i = 1$ となる個数は奇数にならない」
$\Leftrightarrow$「(2.2) の形の全ての節が成り立つ」

例えば、$p = 3$ の場合の線形関係式 (2.1) に対応する CNF 形式の論理式は、次のように $4 = 2^2$ 個の節で書ける：

$$(\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3).$$

3. 「MiniSat」ソルバの出力結果

$A=(a_{ij})$ を $\mathbb{F}_2$ 上の $m\times n$ の行列とする。A に対応する連立方程式

$$\begin{cases} a_{11}x_1 + \cdots + a_{1n}x_n = 0 \\ \qquad\vdots \\ a_{m1}x_1 + \cdots + a_{mn}x_n = 0 \end{cases} \tag{3.1}$$

の i 番目の線形関係式を

$$L^{(i)} : a_{i1}x_1 + \cdots + a_{in}x_n = 0$$

とおく。係数が 1 となる項の個数を p_i（つまり $p_i = |\{j \mid a_{ij} = 1\}|$）とする。第 2. 節で論じたように、$L^{(i)}$ は $q_i = 2^{p_i-1}$ 個の節 $c_1^{(i)}, \ldots, c_{q_i}^{(i)}$ からなる CNF 形式の論理式に変換できる。こららの節と $c^{(0)} := x_1 \vee \cdots \vee x_n$ を付け加えて得られる論理式を

$$C_A := c^{(0)} \wedge \left(\bigwedge_{i=1}^{m} (c_1^{(i)} \wedge \cdots \wedge c_{q_i}^{(i)}) \right)$$

と定義する。つまり論理式 C_A は「連立方程式 (3.1) が成り立ち、かつ、節 $c^{(0)}$ が成り立つ」ことと同値である。

線形代数学より、A がフルランクであることと、連立方程式 (3.1) に自明な解「$(x_1, \ldots, x_n) = (0, \ldots, 0)$」以外の解が存在しないことは、同値である。自明な解は節 $c^{(0)}$ を満たさないので、次の補題が成り立つ。

LEMMA 3.1. 次は同値：

「行列 A はフルランク」 ⇔ 「論理式 C_A はUNSAT」

ただし、UNSAT は論理式 C_A を満たす解が存在しないことを意味する。解がある場合は SAT であると言う。

実際に「MiniSat」ソルバに解かせ “UNSAT” になった場合の出力結果を図 1 と図 2 に載せる。ただし、使用した行列 Z_1 と Z_2 は以下のような行列である。（詳細は省くが、それらは整数論における “多重ゼータ値”[1] という実数が満たす線形関係式から得られる。）

	m	n	p_i の平均
Z_1	7309	959	89.6
Z_2	16936	2028	205.3

表 1： 行列 Z_1 と Z_2 のデータ

1 行につき (p_i の平均)/n の割合で 1 が存在するので、どちらの行列も約 90%は 0 で構成されている。なお、節 $c_j^{(i)}$ の個数は指数的（約 $m \times 2^{(p_i \text{ の平均})}$ 個）に増加するため、MiniSat ソルバに入力する際には、節数が少ない（大体 $O(n^2)$ 個の）論理式 C'_A を使用している。それは以下のようにして得られる：

1. 節 $c^{(0)}$ を含む論理式 C を作成。
2. C を MiniSat ソルバに解かせる。SAT となり解を出力したなら「3」へ、UNSAT なら「4」へ進む。
3. 出力された解を全ての線形関係式に代入し、矛盾が生じたら禁止する節を作成し C に追加。そして「2」に戻る。（注意：実験に使用している行列 Z_1, Z_2 では必ず矛盾が生じるので、生じない場合は今は考えない。）

```
WARNING: for repeatability, setting FPU to use double precision
============================[ Problem Statistics ]=============================
|                                                                             |
|  Number of variables:          959                                          |
|  Number of clauses:          383709                                         |
|  Parse time:                   0.82 s                                       |
|                                                                             |
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
==============================================================================
|       100 |     959   383709 24249658 |   140693      100      8 |  0.002 % |
==============================================================================
restarts              : 2
conflicts             : 137          (153 /sec)
decisions             : 198          (0.00 % random) (221 /sec)
propagations          : 8322         (9269 /sec)
conflict literals     : 920          (57.09 % deleted)
Memory used           : 154.00 MB
CPU time              : 0.897863 s

UNSATISFIABLE
```

図 1: 行列 Z_1 の場合の MiniSat ソルバの出力結果

```
WARNING: for repeatability, setting FPU to use double precision
============================[ Problem Statistics ]=============================
|                                                                             |
|  Number of variables:         2028                                          |
|  Number of clauses:        4850564                                          |
|  Parse time:                  33.27 s                                       |
|                                                                             |
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
==============================================================================
|       100 |    2028 4850564 989747581 |  1778540      100    100 |  0.000 % |
|       250 |    2028 4850564 989747581 |  1956394      250     60 |  0.000 % |
|       475 |    2028 4850564 989747581 |  2152033      475     41 |  0.000 % |
|       812 |    2028 4850564 989747581 |  2367236      812     32 |  0.000 % |
|      1318 |    1984 4850564 989747581 |  2603960     1315     26 |  2.170 % |
==============================================================================
restarts              : 10
conflicts             : 1431         (40 /sec)
decisions             : 2618         (0.00 % random) (74 /sec)
propagations          : 47578        (1340 /sec)
conflict literals     : 35277        (55.13 % deleted)
Memory used           : 3980.00 MB
CPU time              : 35.5036 s

UNSATISFIABLE
```

図 2: 行列 Z_2 の場合の MiniSat ソルバの出力結果

4. 最後に更新された論理式 C を C'_A として終了。

論理式 C'_A は C_A の節の一部分から成るので、C'_A が UNSAT なら C_A も UNSAT になる。

さて、出力結果の以下のデータに着目する：

	conflicts	CPU time (Parse time)
Z_1	137	0.9s (0.8s)
Z_2	1431	35.5s (33.3s)

表 2： 出力結果の観察

「conflicts」は、UNSAT を導くまでに探索した回数（変数に値を代入して、特定の節内の変数の値が全て偽になる矛盾が発生した回数）である。「CPU time」 は計算時間であり、「Parse time」は CNF ファイル（CNF 形式の論理式を記述したテキストファイル）を読み込むのにかかった時間である。また論理式 C'_A を作成し CNF ファイルを出力することも考慮に入れると、計算時間は「CPU time」の 10 ～ 100 倍となる。

行列 A のフルランク性を MiniSat ソルバに解かせた場合の長所短所を以下にまとめる：

【長所】 探索回数が非常に少ない

【短所】 CNF ファイルの生成時間とソルバに対する入力処理時間が非常に長い

特に探索回数の最悪回数が 2^n であることを考えると、今回の探索回数は n 以下となっており極端に少ない数値を示している。これは行列 A に対応する論理式 C_A が、探索を容易にする特別な構造を持っており、MiniSat ソルバがその構造を上手に活用していることを示唆している。

短所を克服し長所を享受するためにはどうしたらよいか。一つの解決方法として、「MiniSat ソルバを使用する」のでは

なく「MiniSat ソルバに実装されている技術要素を使用する」ことに着目する。そうすれば、CNF ファイルを生成しソルバに入力する手間がなくなるので、短所を克服できる。しかし、MiniSat ソルバに実装されている技術要素は、DPLL 手続き、CDCL 手続き、VSIDS 変数選択、バックジャンプ法 、リスタート戦略、部分解のキャッシングなど多岐にわたり [5]、全てを網羅するのは容易ではない。そこで本論文では、次の 2 つの技術要素のみを参考にする：

(i). 変数選択　　(ii). CDCL 手続き

注意点として、実際の MiniSat ソルバでは (i) と (ii) は動的かつ複雑に実装されている。つまり「(i) → (ii) → (i) → ⋯」と状況に応じて交互に処理が行われる。しかし提案アルゴリズムは、(i) の後に (ii) を処理して終了しており、静的に実装されている。また MiniSat では、変遷選択を探索中に動的に変更する VSIDS 変数選択手法が使われているが、本提案手法では行列のランク計算でよく使われるより簡単な変数選択の手法を採用している。従って (i) に関しては、「良い変数選択が存在する」という事実のみを参考にしている。以上のように、実は提案アルゴリズムは、MiniSat ソルバのほんの一部の技術要素しか実際には使っていない。しかしそれでも、十分な計算速度が実現される（第 5. 節参照のこと）。

4. 提案アルゴリズム

4.1 変数選択

行列 $A = (a_{ij})$ のランクこと階数は、行基本変形して得られる行階段行列の段数と等しい。従って階段が多くなるように、列の順番を変更することを考える。連立方程式 (3.1) より、それは変数 x_i の順番を変更することに対応している。

連立方程式 (3.1) の i 番目の線形関係式 $L^{(i)}$ に対して、

$$X^{(i)} := \{x_1^{(i)}, \ldots, x_{p_i}^{(i)}\} \subset \{x_1, \ldots, x_n\}$$

を $L^{(i)}$ に現れる変数の集合とする（つまり $L^{(i)} \leftrightarrow x_1^{(i)} + \cdots + x_{p_i}^{(i)} = 0$）。本論文では次のようにして列の順番を変更する。（なお実際には並列化して実装している。）

【変数選択のアルゴリズム】

1. 空のリスト $\mathcal{X} = [\,]$ を用意。
2. 変数の集合 $X^{(i)}$ を要素数が少ない順に並び替える。
3. $|\mathcal{X}| = n$ ならば「4.」へ、そうでない時は以下を実行。
   i. 先頭の $X^{(1)}, X^{(2)}, \ldots, X^{(k)}$ を
   $$|X^{(j+1)} \setminus (X^{(j)} \cup \mathcal{X})| \leq 3 \qquad (1 \leq j < k)$$
   が成り立つ限り取得する（右辺の 3 は変更可能）。
   ii. $k > 0$ なら次のステップへ進む。$k = 0$ の場合は、全ての $X^{(i)}$ を以下のように更新する：「今までに $\mathcal{X}$ に加えた変数を、$X^{(i)}$ から消去する」。更新後、$X^{(i)} = \phi$ となるものは破棄して、要素数が少ない順に $X^{(i)}$ を並び替える。そして、先頭にある最小の要素数の $X^{(1)}, X^{(2)}, \ldots, X^{(k)}$ を取得する。
   iii. $j = 1, 2, \ldots, k$ に対して、$|X^{(j)} \setminus \mathcal{X}| = 1$ なら $\mathcal{X}$ を更新：
   $$\mathcal{X} \leftarrow X^{(j)} \cup \mathcal{X}.$$
   ただし $X^{(j)}$ の要素は $\mathcal{X}$ の先頭に追加。また $j = 1$ の時は条件に関わらず必ず追加する。
   iv. $\mathcal{X}$ の更新で使用した $X^{(j)}$ を破棄して、「3.」へ戻る。
4. $\mathcal{X}$ を $[x_{r_1}, x_{r_2}, \ldots, x_{r_n}]$ とおく。行列 A の列の順番を
   $$r_i \to i \qquad (i = 1, \ldots, n)$$
   と変更する。
5. 階段行列になるように行をソートして終了。

一つ、上記のアルゴリズムから得られる結果の例を示す：

$$A = \begin{pmatrix} 1&1&1&1&0&1 \\ 0&1&1&0&0&0 \\ 0&1&1&1&1&1 \\ 1&1&1&1&1&1 \end{pmatrix} \quad \to \quad A' = \begin{pmatrix} 1&1&1&1&1&1 \\ 1&1&0&1&1&1 \\ 0&1&1&1&1&1 \\ 0&0&0&0&1&1 \end{pmatrix}$$

例 1：$\mathcal{X} = [x_1, x_6, x_5, x_4, x_3, x_2]$ となり、 列の順番を $(1, 6, 5, 4, 3, 2) \to (1, 2, 3, 4, 5, 6)$ に変え、行をソートした例

上記アルゴリムのイメージを説明する。階段行列は、「左下に 0、右上に 1」が多い方が階段数が多くなる。$X^{(i)}$ を要素数が少ない順に並び替えることにより、$\mathcal{X}$ の後方には、項数の少ない線形関係式に現れやすい変数が、格納されるようにしている。また階段が生じるように条件を付けて $\mathcal{X}$ を更新している（「ステップ 3-iii」参照）。なお、$\mathcal{X}$ の更新に使用する $X^{(i)}$ の個数を閾値で制限する「ステップ 3-i」がないと、後の処理において全ての $X^{(i)}$ に関して計算することになり、非常に時間がかかる。（閾値で制限する手法は [7] に紹介されている頻出パターン発見アルゴリズムに影響を受けている。）

4.2 探索手続き

SAT 問題（CNF 形式の論理式）に対して、DPLL 手続きという単位伝播（unit propagation）を利用した探索的アルゴリズムが開発されている。それは、適当に（あるいはランダムに）変数に真偽値を割り当てて、充足する解を探す。発見できたら “SAT” を出力し、逆に全ての場合の変数割り当てで発見できなかったら “UNSAT” を出力する。単位伝播とはその探索の際に使われる技術であり、単位節（変数が一つの節「x_i または $\neg x_i$」）があった場合、その変数 x_i の値が確定できることを利用し、計算時間を大幅に減少させている。CDCL 手続きとは、学習節（動的に学習して得られた新たな節）を追加する処理が DPLL 手続きに加わったアルゴリズムのことである（詳細は [5] を参照）。

行列 A' を階段行列とする。A' で角ができていない列番号の数列を $\mathcal{K} = (k_1, \ldots, k_g)$ $(k_i < k_{i+1})$ とおく。例えば例 1 の場合「$\mathcal{K} = (3, 4, 6)$」となる。また便宜上 $k_0 = 0$ と約束する。

入力 A' に対して “FULL” か “UNFULL” を出力する、CDCL 手続きを参考にした提案アルゴリズムを以下に書く。（FULL が「フルランクであること」、UNFULL が「フルランクでないこと」を意味する。）自明な事実として、線形関係式

$$L^{(i)} : x_1^{(i)} + x_2^{(i)} + \cdots + x_p^{(i)} = 0$$

があった時、$(p-1)$ 個の変数 $x_2^{(i)}, \ldots, x_p^{(i)}$ に値が割り当てられたならば、残り 1 つの変数 $x_1^{(i)}$ の値は確定する。この事実が SAT 問題における単位伝播と同じ役割をする。下記のアルゴリズムに現れる線形関係式は、変数への値の割り当ての仕方により、全て単位伝播が可能な状態になっていることに注意する。なお「ステップ 2-iv」の新たに線形関係式 L を計算している部分が、CDCL 手続きにおいて学習節を導出している部分に相当すると考えられる。

【探索のアルゴリズム】

1. $a = 1$ とする。
2. $a = g + 1$ なら「3」へ。そうでない時は、

$$x_{k_a} = 1, \qquad x_{k_a+1} = \cdots = x_n = 0$$

を割り当てて、以下を実行。

   i. $k = k_a - 1$ とする。
   ii. x_k を含みかつ x_i $(i < k)$ を含まない線形関係式のリスト $\mathcal{L} = [L^{(1)}, \ldots, L^{(s)}]$ を取得。
   iii. $\mathcal{L} = \phi$ の時 “UNFULL” を出力して終了。そうでない時は、$L^{(i)}$ の単位伝播により確定される変数 x_k の値 $y^{(i)}$ を取得。
   iv. $y^{(u)} \neq y^{(v)}$ となる $u < v$ が存在しない時、x_k に $y^{(1)}(= y^{(2)} = \cdots = y^{(s)})$ を割り当て $k \leftarrow k-1$ として、「ii」へ戻る。存在する時は、$L^{(u)}$、$L^{(v)}$ と他の線形関係式を適当に組み合わせて基本変形をし、以下の条件を満たす線形関係式 Lを計算する：
      - 変数 x_{k_a} を含むが $x_i (i < k_a)$ を含まない

      （証明は紙数の都合上省くが、上記のような L を必ず計算できることが線形代数学から導ける。）
   v. $a \leftarrow a + 1$ として「2」へ戻る。

3. “FULL” を出力して終了。

行列 A' は A の行と列番号を変換したものなので、これら二つのランクは同じ値である。従って、A' に関する出力結果から A についてのフルランク性がわかる。

5. 実験結果

比較実験の結果を報告する。実験で使用した環境は、CPU が 2 つの Intel Xeon E5-2670 2.60GHz、RAM が 512GB、OS が CentOS 6.7 である。また、多重ゼータ値 [1] の線形関係式から得られるフルランクの行列を使用している。

表 3 では （single と parallel の）ガウスの消去法との計算速度を比較している。本提案手法の方が上回っていることが確認される。前処理を施しているため、各々が実際に計算している行のサイズは多少異なるが、大きくは変わらない。なおコードは Python 3.4 を（提案手法はさらに Cython も）使用している。

論文 [4] では、並列化したガウスの消去法を Python ではなく C 言語で実装している。その論文で使用している行列と本研究で使用している行列はかなり異なるが、基本的には行基本変形で移りあえることが理論的に保証されている。従ってランクの観点では本質的な差はないので、その比較を表 4 に載せることにする。ただし下記のように大きく異なる点があるので厳密な比較はできないことを付記する：論文 [4] では、

- フルランク性ではなく具体的なランクの値を計算している。
- 体 $\mathbb{F}_2$ ではなく体 $\mathbb{F}_{16381}$、$\mathbb{F}_{31991}$ を使用している。
- 前処理でサイズが約半分の正方行列に帰着させている。
- 実験環境が異なる：「8CPU: Intel Xeon X5355 2.66GHz (8 core)、RAM: 32GB、OS: Linux」。

サイズ (m 行, n 列)	ガウス（S）	ガウス（P）	提案手法
(18973, 4075)	4m30s	1m	**16s**
(40964, 8164)	36m	4m	**1m2s**
(87998, 16347)	5h25m	26m	**3m30s**
(188421, 32719)	-	3h	**12m30s**

表 3 ：ガウスの消去法「single(S) と parallel(P)」との速度比較
（ガウスの消去法は Python 3.4 を、提案手法は加えて Cython を使用）

サイズ (m 行, n 列)	論文 [4]	提案手法
(8192, 8192), (**87998**, **16347**)	2m	**3m30s**
(16384, 16384), (**188421**, **32719**)	13m	**12m30s**
(32768, 32768), (**401095**, **65471**)	1h18m	**54m**
(65536, 65536), (**851972**, **130986**)	9h	**3h33m**
(131072, 131072), (**1006556**, **262030**)	67h	**5h27m**

表 4 ：論文 [4] との比較
（論文 [4] は C 言語、提案手法は Python3.4 と Cython を使用）

単純に比較はできないが、提案手法の方が行列のサイズが大きくなっても、計算時間の増加度が小さい良い特徴があると考えられる。実際、行列のサイズが約 4 倍（行と列のサイズがそれぞれ約 2 倍）になるごとに、論文 [4] では約 6~8 倍、提案手法が約 3~5 倍で計算時間が増加している。なお、最下段の「(**1006556**, **262030**)」の場合の提案手法の倍率が低いのは、メモリの制限のため行サイズを減らしたからである。（行を減らしてフルランクなら元の行列でもフルランクである。本実験ではフルランクの行列を使用しているため、行を減らしても問題はない。）

6. まとめ

本論文では、SAT ソルバの要素技術を参考にした、行列のフルランク性を判定する手法を提案した。参考にしている技術はほんの一部（CDCL 手続き）であり、さらなる改良が必要である。しかしそれでも、整数論の多重ゼータ値 [1] から得られる特殊な行列においては、ガウスの消去法よりは速くなることを実験で確認した。任意の行列でも同じような結果になるかは今後の研究で解明する予定である。

参考文献

[1] 荒川恒男, 金子昌信, **多重ゼータ値入門**, Math-for-Industry (MI) Lecture Note Series, 23 巻, 2008.

[2] N. Eén, N. Sörensson, *An Extensible SAT-solver*, in International conference on theory and applications of satisfiability testing, pp.502–518, Springer 2003.

[3] 井上克巳, 田村直之, *SAT* **ソルバーの基礎**, 人工知能学会誌, 25 巻, pp.57-67, 2010.

[4] M. Kaneko, M. Noro, K. Tsurumaki, *On a conjecture for the dimension of the space of the multiple zeta values*, Software for algebraic geometry **148** (2008), 47–58.

[5] 鍋島英知, 栄剛秀, **高速** *SAT* **ソルバーの原理**, 人工知能学会誌, 25 巻, pp.68-76, 2010.

[6] T. Sonobe, *Learnt Clause Sharing in Parallel SAT Solvers by Using Community Structure* , The 30th Annual Conference of the Japanese Society for Artificial Intelligence, 2016.

[7] T. Uno, H. Arimura, *An Introduction to Frequent Pattern Mining - Itemsets to Graphs* - , The 22th Annual Conference of the Japanese Society for Artificial Intelligence, 2008.