F-039

命題論理式の解の一様サンプリングの改善
Improving Near-Uniform Sampling of Boolean Satisfiability Solutions

中島　祐輝[1)]　戸田　貴久[2)]
Yuki Nakajima　Takahisa Toda

1　概要

充足可能性問題 SAT とは与えられた命題論理式が真となるような解が存在するか判定する問題である. これまでの主な SAT 応用は複数存在する解の中から解を 1 つだけ見つけることを基本にしていたが, 近年では SAT サンプリングという, 解を 1 つだけ求めるのではなく一様ランダムにサンプリングする方法が研究されている. 本論文では SAT サンプリングの従来手法に対し, 解のサンプリングの一様性を改善する手法を提案する. そして従来手法との比較実験を行い, サンプリングの一様性が改善されることを確認する.

2　研究の背景

充足可能性問題 SAT とは与えられた命題論理式が真となるような割り当て (解) が存在するか判定する問題である. この SAT を解くためのプログラムを SAT ソルバーといい, 現在も活発に開発されている. 現実の問題でも SAT に変換できれば SAT ソルバーで解を求めることができる. スケジューリングやプランニングなどの問題も SAT 符号化することにより SAT ソルバーで解を求めることができる.

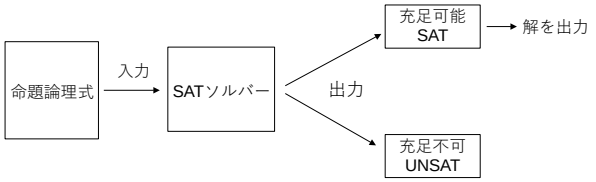

図 1　SAT と SAT ソルバーの関係

これまでの主な SAT 応用は複数存在する解の中から解を 1 つだけ見つけることを基本にしていた. どの解を見つけるかは SAT ソルバーの探索アルゴリズムによって決まる. そして近年では **SAT サンプリング**という, 解を 1 つ求めるのではなく一様ランダムにサンプリングする方法が研究されている. これまでに様々な SAT サンプリング手法が考案されてきたが, いずれの手法のサンプリング精度も十分なものではない. 従来の手法では問題を SAT に変換 (SAT 符号化) するときに導入する**補助変数**を**主要変数**と区別せずにサンプリングを行っている. 補助変数は符号化のために導入する変数で, 元の問題の解を表す主要変数の数と比べると実際に大幅に多いため, 区別せずにサンプリングを行うとサンプリング精度に影響を及ぼす可能性があると考える.

そこで補助変数がサンプリング精度に及ぼす影響に着目し, 本研究では主要変数と補助変数を区別することにより補助変数の影響を排除したサンプリング方法を提案する. そして従来手法と提案手法でサンプリングの一様性の比較実験を行う. 比較実験で使用するサンプリング対象の問題は, ランダムに生成した連立不等式とした. 不等式を選んだ理由については, 不等式は様々なアルゴリズムや制約において頻出する式であるためである.

本論文では, 第 3 節で本研究で必要になる用語について説明をする. 第 4 節では SAT サンプリングの従来手法について, 第 5 節では提案手法について説明をする. そして第 6 節では実験の方法, 実験の結果とその結果から考察を行う. 第 7 節では本研究のまとめを行う.

3　基礎知識

本節では本研究で必要になる用語について説明する.

3.1　充足可能性問題

充足可能性問題 (Satisfiablility;SAT) とは与えられた命題論理式が真となるような割り当て (解) が存在するか判定する問題である. 命題論理式とは命題変数あるいは定数 0,1 を論理演算子 AND（論理積,∧）,OR（論理和,∨）,NOT（否定,¬）で結んで得られる式を指す. そして変数あるいはその否定を**リテラル**と呼び, リテラルを論理和で結んだものを**節**という. 命題変数は **1(真)** または **0(偽)** の値をとる. この命題論理式に現れる命題変数に 1 または 0 を割り当てることを**真偽値割り当て**という. そして与えられた命題論理式が真となる真偽値割り当てが存在するとき, **充足可能**という. 反対に命題論理式が真となる真偽値割り当てが存在しないとき, **充足不能**という. また, 一般的に SAT 問題において命題論理式は**連言標準形 (conjunctive normal form; CNF)** で表される. 連言標準形は節を論理積で結合した命題論理式である.

そして SAT を解くプログラムを **SAT ソルバー**という. SAT ソルバーを使うことで与えられた命題論理式が充足可能か充足不能かどうかを判定し, 充足可能ならば充足可能解を 1 つ出力する.

3.2　XOR 制約

現在の SAT サンプリングは XOR 制約を用いたサンプリング手法が主流となっている. XOR 制約というのは命題変数や定数を排他的論理和（XOR）で結んで得られる式である. XOR 演算子は $\oplus$ で表す.XOR 演算子を使うと XOR 制約は式 (1) のように表される.

$$x_1 \oplus x_2 \oplus \ldots \oplus x_k \tag{1}$$

XOR 制約にて同じ変数が奇数個現れる場合, それらをまとめて 1 つに置き換えても置き換える前の演算結果と同値となる. 同様に定数 1 が奇数個現れる場合もまとめて 1 つに置きかえても同値となる.

1）　電気通信大学大学院　情報理工学研究科
E-mail:nakajima@disc.lab.uec.ac.jp
2）　電気通信大学大学院　情報理工学研究科
E-mail:toda@disc.lab.uec.ac.jp

一方で同じ変数が偶数個現れる場合にはそれらを削除しても, 削除する前の演算結果と同値となる. 同様に定数 1 が偶数個現れる場合もそれらを削除しても同値となる.

また XOR 制約の演算結果は式の中で 1 となる変数, 定数の数によって決まる. つまり定数 0 の有無は XOR 制約の演算結果に影響を及ぼさないため無視, 削除できる. その結果 $x_1, ..., x_k$ はすべて異なる変数として XOR 制約は式 (2) のいずれかに表される.

$$\begin{cases} x_1 \oplus x_2 \oplus ... \oplus x_k \\ x_1 \oplus x_2 \oplus ... \oplus x_k \oplus 1 \end{cases} \tag{2}$$

3.3 SAT 符号化

現実の問題も SAT に変換することにより SAT ソルバーで解を求めることができる. この SAT に変換することを **SAT 符号化**といい, 直接符号化, 順序符号化 [1] などの様々な手法がある. 直接符号化は整数変数 x が値 i を取ることを表す命題変数 $p_{x=i}$ を導入することで, 整数変数 x を符号化する. 順序符号化は整数変数 x が値 $x \leq i$ であることを表す命題変数 $p_{x \leq i}$ を導入することで, 整数変数 x を符号化する. また整数変数を変換する以外にも不等式を SAT に変換する手法もある. そこで不等式を変換する手法の 1 つである Sinz の符号化 [2] を用いて SAT 符号化について説明する. 式 (3) のような不等式がある時,Sinz の符号化を用いることで $1 \leq j < n - r$ かつ $1 \leq k \leq r$ を満たす全ての j, k に対して式 (4) の形の制約, そして $1 \leq j \leq n - r$ かつ $0 \leq k \leq r$ を満たす全ての j, k に対して式 (5) の形の制約として符号化される.

$$x_1 + ... + x_n \leq r \qquad (x_i = \{0, 1\}) \tag{3}$$

$$(\neg y_{j+(k-1)(n-r)} \vee y_{j+(k-1)(n-r)+1}) \tag{4}$$

$$(\neg x_{j+k} \vee \neg y_{j+(k-1)(n-r)} \vee y_{j+k(n-r)}) \tag{5}$$

この時 $k = 0$ ならば $\neg y_{j+(k-1)(n-r)}$ は省略,$k = r$ ならば $y_{j+k(n-r)}$ は省略される. 実際に式 (6) を符号化した結果は式 (7) になる.

$$x_1 + x_2 + x_3 + x_4 \leq 2 \qquad (x_i = \{0, 1\}) \tag{6}$$

$$\begin{gathered} (\neg y_1 \vee y_2) \wedge (\neg y_3 \vee y_4) \wedge (\neg x_1 \vee y_1) \wedge \\ (\neg x_2 \vee \neg y_1 \vee y_3) \wedge (\neg x_3 \vee \neg y_3) \wedge (\neg x_2 \vee y_2) \wedge \\ (\neg x_3 \vee \neg y_2 \vee y_4) \wedge (\neg x_4 \vee \neg y_4) \end{gathered} \tag{7}$$

式 (7) を見ると命題変数 x_i に加えて, 命題変数 y_i が導入されていることが確認できる. 式 (7) に現れる命題変数 x_i は元の問題である式 (6) に現れる変数と一対一に対応しており, **主要変数**と呼ぶ. 一方, 式 (7) に現れる命題変数 y_i は式 (6) には存在しない. これは符号化のために導入された変数で**補助変数**と呼ぶ. Sinz の符号化では式 (3) のような不等式を符号化すると主要変数が n 個, 補助変数が $(n - r)r$ 個生成される.

4 従来手法

本節では命題論理式の解をサンプリングする従来手法について説明する.

4.1 XORSample

命題論理式の解をサンプリングする（SAT サンプリング）手法の 1 つである Gomes ら [3] によって提案された **XORSample** について説明する. XORSample の処理の流れをフローチャートとして図 2 に示す.

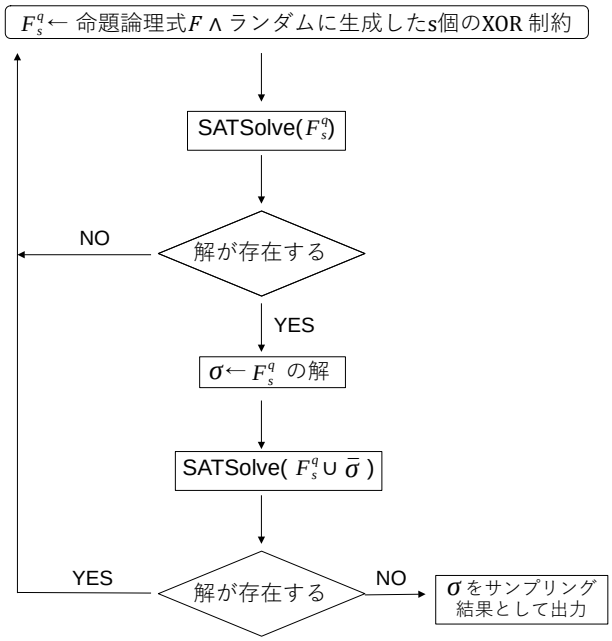

図 2 XORSample の処理の流れ

XORSample の基本的な考え方は命題論理式 F に対してランダムに XOR 制約を加えて問題 F_s^q を生成し,SAT ソルバーで解く. その時, 解がちょうど 1 つだけ存在すれば, それをサンプリング結果として出力する. 解がちょうど 1 つだけ存在するかどうかの判定方法は,F_s^q を SAT ソルバーで解いて解 σ を得られた場合, それを否定した $\bar{\sigma}$ を F_s^q に加えて, もう一度 SAT ソルバーで解くことにより, 解が存在しなければ先ほど得られた解 σ だけが存在することになる. 解が 1 つもない, もしくは 2 つ以上ある場合は解が 1 つになるまで XOR 制約の再生成と SAT ソルバーによる求解を繰り返す. ここで解を否定したもの $\bar{\sigma}$ とは, 解 σ を構成するリテラルの符号を反転したものを指す. 解を否定したもの $\bar{\sigma}$ を問題 F に加えて SAT ソルバーで解くことにより, 解 σ は制約を満たさなくなるので他に解が存在するかどうか判定ができる. XORSample のアルゴリズムの詳細は Algorithm1 に擬似コードで示す.

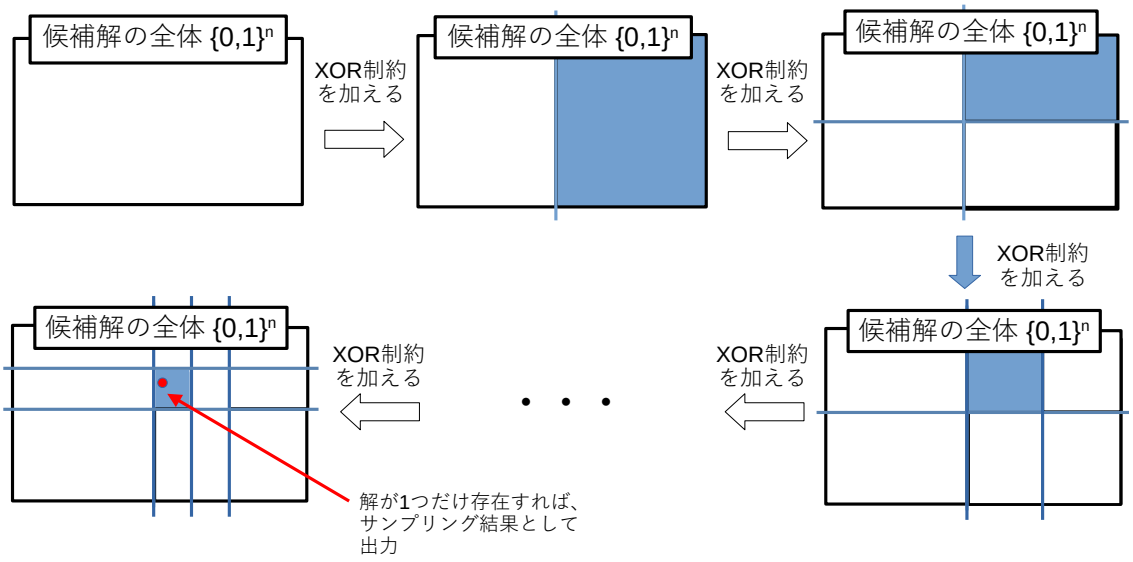

図 3　XOR 制約をつけた時の解空間

Algorithm 1 XORSample

Params: 変数を選択する確率 q, XOR 制約の個数 s
Input : CNF 式 F
Output: F の解

1: $iterationSuccessful \leftarrow$ FALSE
2: **while** $iterationSuccessful$ = FALSE **do**
3: $Q_s \leftarrow$ s 個のランダム XOR 制約
4: $F_s^q \leftarrow F \cup Q_s$
5: result $\leftarrow$ SATSolve(F_s^q)
6: **if** (result = TRUE) **then**
7: $\sigma \leftarrow$ SATSolve(F_s^q) で得た解
8: $F' \leftarrow F_s^q \cup \overline{\sigma}$ //解の否定を加える
9: result' $\leftarrow$ SATSolve(F')
10: **if** (result' = FALSE) **then**
11: $iterationSuccessful$ = TRUE
12: **return** σ
13: **end if**
14: **end if**
15: **end while**

XORSample では事前に 2 つのパラメータ q,s の値を決める必要がある. パラメータ q は XOR 制約を生成するときに各変数を選択する確率. パラメータ s は XOR 制約の個数である. パラメータ q の値について [3] では q=0.5 と設定している. またパラメータ s の値は 2^s が解の総数に近い値になることが望ましいとしている.

また類似した手法として XORSample'[3] がある. XORSample の場合は XOR 制約を加えた後の解空間に解が 1 つだけ存在する場合, その解をサンプリング結果として出力するが,XORSample' の場合は XOR 制約を加えた後の解空間に解が複数個存在することを許容し, その場合複数の解の中からランダムに 1 つを選んでサンプリング結果として出力する. この手法でもサンプリングを行う前にパラメータ q,s の値を決める必要がある. パラメータ q の値は XORSample と同様に q=0.5 が設定されている.

Chakraborty ら [4] によると実験により XORSample' のほうが XORSample より優れたパフォーマンスを得られたと報告されている.

4.2　XOR 制約

なぜ XOR 制約を加えて SAT ソルバーで解くことによってサンプリングが行えるのかについて説明をする. 1 つの XOR 制約を加えると解空間はほぼ同じサイズの 2 つの部分にランダムに分割されることが期待される. 従って XOR 制約を s 個加えることで解空間は $1/2^s$ になることも同様に期待できる. そして XOR 制約を s 個加えた時の解空間に解が 1 つだけ存在すれば, それをサンプリング結果として出力する. つまりランダムに生成した XOR 制約を問題に加えることで解空間もランダムに縮小するため, そこから解をサンプリング結果として出力することで一様ランダム性の高いサンプリングを行うことができる. これを図 3 に示す.

そしてこの XOR 制約がハッシュの考えに関係する. XOR 制約は必ず真または偽となる. つまり 1 個の XOR 制約を {0,1} を出力する関数とみなすことができる. よって問題を構成する命題変数の個数 n 個のとき, s 個のランダムに生成した XOR 制約を加えることにより関数 $h : \{0,1\}^n \rightarrow \{0,1\}^s$ をランダムに決定することに相当する.

4.3　従来手法の問題点

従来手法ではサンプリングを行うときに生成する XOR 制約は主要変数と補助変数から生成される. 補助変数は符号化のために導入される変数であり,1 つの主要変数の真偽値割り当てに対して複数の補助変数の真偽値割り当てが存在する場合がある. そのため主要変数と補助変数を含んで XOR 制約を生成するとサンプリングされる解に偏りが生じ, サンプリングの一様性に影響を及ぼすと考える.

5　提案手法

本章では XORSample でサンプリングを行う際に主要変数と補助変数を区別することにより, 補助変数の影響を排除して解のサンプリングを行う手法を提案する.

5.1 提案手法の説明

提案手法について説明をする. 提案手法では XOR 制約の生成方法を工夫することにより補助変数の影響を排除することを図る. 従来手法では主要変数と補助変数から XOR 制約を生成している. そこで XOR 制約を構成する変数を主要変数のみに限定することで補助変数の影響を排除する.

また従来手法では XOR 制約を加えた後に SAT ソルバーで解を求め, その解を否定したものを問題に加えてもう一度解くことにより解が 1 つだけ存在するか判定を行っていた. しかしその方法を提案手法でも用いると,1 つの主要変数の真偽値割り当てに対して補助変数の真偽値割り当てが複数存在する解はそれぞれ異なる解と判定されてしまう. 図 4 の例を用いて説明する.

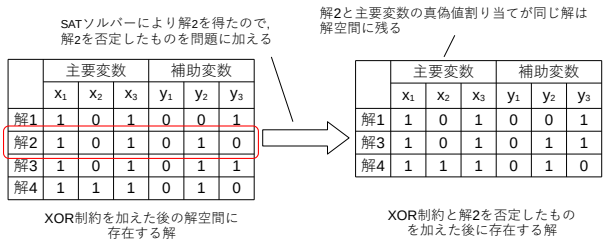

図 4 XOR 制約を主要変数から構成した時に従来手法で唯一解か判定するときの問題点

図 4 の左の表は問題 F に XOR 制約を加えた後の解空間に存在する解の一覧で, 右の表は解 2 を否定したものをさらに加えた時の解空間に存在する解の一覧である. 左の表では XOR 制約を加えた後の問題に解が 4 つ存在する. これら 4 つの解のうち 3 つの解は主要変数の真偽値割り当ては同じだが, 補助変数の真偽値割り当てはそれぞれ異なる. 従来手法では XOR 制約を加えた問題の解を SAT ソルバーで得たら, その解を否定したものを加えて解きなおす. 図 4 では解 2 を否定したものを加えて解きなおすが, その時の解空間には主要変数の真偽値割り当てが解 2 と同じなのにも関わらず解 1 と解 3 は解空間に残ってしまう.

よって 1 つの主要変数の真偽値割り当てに対して補助変数の真偽値割り当てが複数存在する解をそれぞれ同じ解と判定するためには否定した解をそのまま使うのではなく工夫する必要がある. そこで提案手法では, 否定した解の変数全てを加えるのではなく, 否定した解の主要変数のみを加えるよう変更を加える. そうすることにより主要変数の真偽値割り当てが同じだが補助変数の真偽値割り当てが異なる解はいずれも同じ解として判定されるため, 主要変数の真偽値割り当てが同じ解は解空間から消える. これを図 5 に示す. 図 5 の左の表は問題 F に XOR 制約を加えた後の解空間に存在する解の一覧である. そして SAT ソルバーで解 2 を得て, 解 2 の否定した主要変数のみを問題 F に加えた時の解空間に存在する解は右の表になる. 解 2 だけでなく主要変数の真偽値割り当てが同じである解 1, 解 3 も同時に解空間から削除することができ, 解が 1 つだけ存在するか判定を行うことができる.

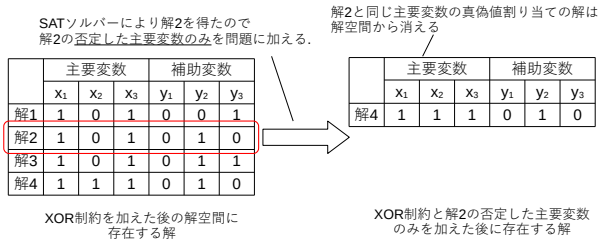

図 5 唯一解の判定に否定した主要変数のみを加えた場合

もし主要変数の真偽値割り当ての種類が複数ある場合は別の解として扱われるので, その場合は従来手法と同様に XOR 制約を再生成する.

5.2 提案手法の疑似コード

提案手法の疑似コードを Algorithm2 に示す. 従来手法の Algorithm1 との変更点は赤字にしてある.

Algorithm 2 提案手法

Params: 変数を選択する確率 q, XOR 制約の個数 s
Input : CNF 式 F
Output: F の解

```
1:  iterationSuccessful ← FALSE
2:  while iterationSuccessful = FALSE do
3:      Q_s ← 主要変数のみで構成された s 個のランダム XOR 制約
4:      F_s^q ← F ∪ Q_s
5:      result ← SATSolve(F_s^q)
6:      if (result = TRUE) then
7:          σ ← SATSolve(F_s^q) で得た解
8:          σ′ ← 主要変数に限定した σ
9:          F′ ← F_s^q ∪ σ′ //否定した解の主要変数を加える
10:         result' ← SATSolve(F')
11:         if (result' = FALSE) then
12:             iterationSuccessful = TRUE
13:             return σ
14:         end if
15:     end if
16: end while
```

6 実験結果と考察

本章では従来手法と提案手法の比較実験を行い, その結果に対する考察を行う.

6.1 実験の設定

性能評価を行うための, 従来手法と提案手法の比較実験の設定について説明する. 本実験では従来手法と提案手法を用いてサンプリング実験を行い, サンプリング結果の一様性を標準偏差を用いて数値化し比較する. 比較実験に使用する手法を以下にまとめる.

- 従来手法: Gomes らによって提案された XORSample
- 提案手法: 補助変数の影響を排除した XORSample

比較実験を行ううえで, サンプリングに使用する問題を選択する必要がある. 選択する問題については, 問題の規模が大きくなってしまうとサンプリングに時間がかかってしまうので比較実験を行うのに十分な回数のサン

プリングを行うのが困難になってしまう. そこで本実験では連立不等式の解をサンプリングする問題とする. 連立不等式を選択した理由については以下のものがある.

- 不等式は様々な問題に表れる式であるため解をサンプリングするのに意義がある
- 規模の小さい問題から大きい問題まで様々な問題を生成できる

実際に実験に使用する不等式はランダムに生成したものを使うこととした. サンプリング精度の比較実験を行うためにランダムに生成した連立不等式を式 (8) に示す.

$$\begin{cases} x_1 + x_2 + x_5 + x_6 + x_8 + x_{10} \geqq 3 \\ x_1 + x_2 + x_4 + x_5 + x_9 \geqq 2 \\ x_1 + x_2 + x_4 + x_6 + x_9 \leqq 3 \end{cases} \quad (x_i = \{0, 1\}) \tag{8}$$

この連立不等式の解は 432 個存在することを確認した. ここで連立不等式の解とは以下のように不等式に現れる x_1 から x_{10} までの変数の割り当てを指している.

$$(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}) = (1, 0, 1, 0, 1, 1, 1, 1, 1, 0) \tag{9}$$

そして式 (8) を Sinz の符号化を用いて符号化した. その結果, 変数の数は 31 個, 節の数は 40 個の命題論理式が生成された. 変数 31 個中の主要変数と補助変数の数の内訳は主要変数は 10 個, 補助変数は 21 個となっている. この命題論理式には解が 13860 個存在することを確認した. ここで命題論理式の解とは以下のように命題論理式に現れる x_1 から x_{31} までの変数の割り当てを指している.

$$(x_1, ..., x_{10}, x_{11}, ..., x_{31}) = (1, ..., 0, 0, ..., 0) \tag{10}$$

この時 $x_1, ...x_{10}$ が主要変数となり, $x_{11}, ...x_{31}$ が補助変数となっている.

次に各手法でのパラメータの値を決定する. パラメータ q の値は両手法ともに q=0.5 とした. 従って, 従来手法では XOR 制約の長さの期待値は 32/2 = 16 となる. そして提案手法では XOR 制約の長さの期待値は 11/2 ≈ 6 となる. 次にパラメータ s の値を決定する. パラメータ s の値は 2^s が解の総数に近い値になることが望ましいとしている. 従来手法では XOR 制約を主要変数と補助変数から構成するので, 命題論理式の解の総数を考えると 13860 個となる. そのため従来手法のパラメータ s の値は s=14 とした. 一方で提案手法は XOR 制約を主要変数のみから構成する. 主要変数の解の総数は連立不等式の解の総数と等しいため 432 個となる. よって提案手法のパラメータ s の値は s=9 とした.

次に性能評価の方法について説明を行う. 初めに従来手法, 提案手法でサンプリングを行い, 各解が何回サンプリングされたか記録する. そして性能評価を行うために, 各解のサンプリング回数の平均を表す数値 *uniform* を求める. *uniform* の値の求め方は式 (11) となる.

$$uniform = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{11}$$

ここで n は解の総数,x_i は解 i がサンプリングされた回数である. この *uniform* の値を平均値として, 各手法での標準偏差を求めて比較を行う. 具体的な標準偏差の計算方法については式 (12) に示す.

$$s = \sqrt{\sum_{i=1}^{n} (x_i - uniform)^2} \tag{12}$$

そして本実験を行う環境を以下に示す.

- OS: CentOS Linux 8.2
- 主メモリ: 32GB
- CPU: Intel Xeon CPU E5-2609 v4 1.70GHz

またいずれの手法でも SAT ソルバーは Cryptominisat 5.8.0[5] を利用する.

6.2 実験の結果と考察

式 (8) を符号化して得られた命題論理式の解を従来手法と提案手法でサンプリング回数を 10 万回に設定してサンプリングを行った. 各手法での標準偏差, サンプリングにかかった時間を表 1 に示す. また従来手法でのサンプリング結果を図 6, 提案手法でのサンプリング結果を図 7 に示す. 各図の横軸は不等式の解を表す. 縦軸はその解が何回サンプリングされたかを表す. ここで縦軸は対数スケールで与えている.

表 1　各種手法でサンプリング実験を行った結果

	従来手法	提案手法
標準偏差	382.7	49.5
10 万回のサンプリングに要した時間の合計（分）	91	100

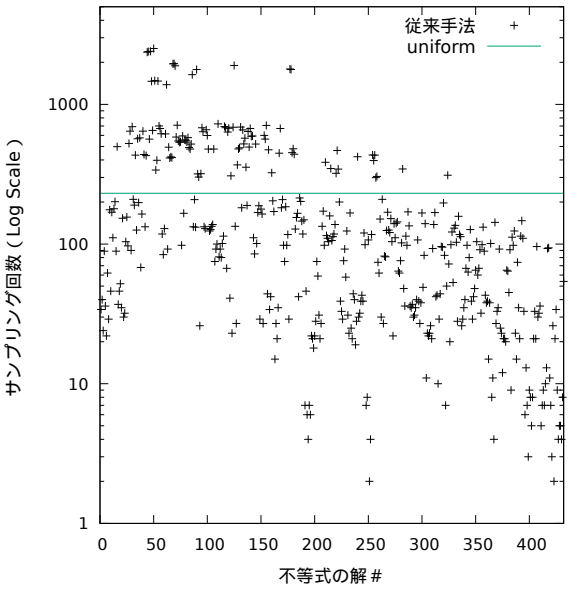

図 6　従来手法でのサンプリング結果

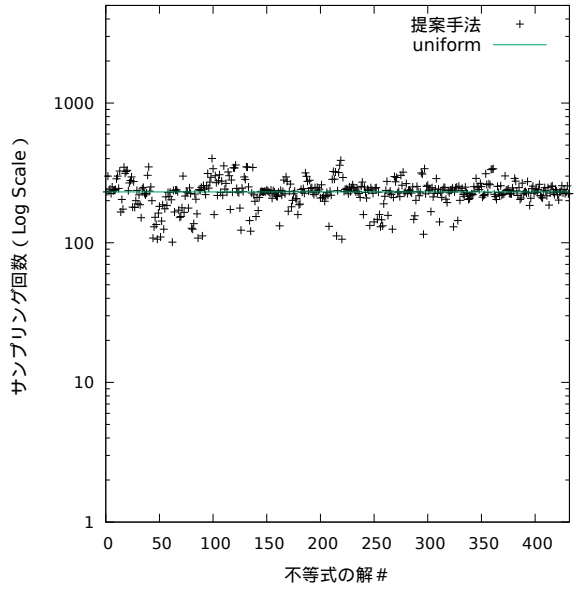

図 7　提案手法でのサンプリング結果

従来手法と提案手法を比較する. 表 1 を見ると提案手法のほうが大幅に標準偏差が小さいことが確認できた. この原因について主要変数の真偽値割り当てが決まっても補助変数の真偽値割り当てが一意に決まらない解が存在したことが原因と考える. 実際, 今回のサンプリング実験に使用した不等式に解は 432 個存在するのに対して,SAT 符号化して得られた命題論理式には解が 13860 個存在する. そして命題論理式の解を確認すると, 主要変数の真偽値割り当てが決まると補助変数の真偽値割り当てが一意に決まる解が存在するのに対して, 主要変数の真偽値割り当てが決まると補助変数の真偽値割り当ての種類が 300 個以上存在する解も確認できた. そのため従来手法のように XOR 制約を主要変数と補助変数から構成すると, サンプリングしたい母集団を考えた時に, サンプリング結果に偏りが生じてしまうため図 6 にあるように *uniform* から大きく離れた回数サンプリングされてしまう解が現れたと考える.

また今回の実験で使用した SAT ソルバーは XOR 制約が長くなると解を求めるのに時間がかかるという問題点があった. 今回使用した問題は規模が小さかったため,XOR 制約の長さの期待値が提案手法の場合は 16, 従来手法の場合は 6 でそれほど差がなかったため, 提案手法と従来手法でサンプリングにかかる時間にそれほど差は生じなかった. しかし問題の規模が大きい場合は符号化により多くの補助変数が導入される. そして従来手法だと主要変数と補助変数から XOR 制約を構成するので,XOR 制約が長くなってしまいサンプリングに時間がかかってしまう. 一方で本研究のアイデアである補助変数の影響を排除するために主要変数のみで XOR 制約を構成することで, XOR 制約の長さが短くなるためサンプリングにかかる時間の減少が期待できる.

7　まとめ

本研究では, 補助変数の影響を排除してサンプリングすることによりサンプリング結果の一様性を改善する方法を提案した. そして連立不等式の解を従来手法と提案手法でサンプリングを行い, サンプリング結果の一様性を比較した. その結果, 主要変数の真偽値割り当てが決まっても補助変数の真偽値割り当てが一意に決まらない解が存在する場合に提案手法でサンプリングの一様性が大幅に改善したことが確認できた. また補助変数の影響を排除するために主要変数のみで XOR 制約を構成することにより,XOR 制約の長さの期待値が小さくなるので, 問題の規模が大きくなればサンプリングにかかる時間が減少することも期待できる.

今回の比較実験では 1 つの連立不等式に対して従来手法と提案手法で解のサンプリングをしたときの一様性が改善されたかを確認した. 今後は連立不等式以外の問題に対しても同様の比較実験を行い, 同様の効果が表れるかを確認したい.

また,XORSample はハッシュに基づくサンプリング手法として初期に提案された手法であり, 最新の SAT サンプリングの手法として ApproxMC[6] などが知られている. このような手法に対しても本研究のアイデアを適応して, 同様の性能評価を行いたいと考えている.

謝辞

本研究は JSPS 科研費 17K17725 の助成を受けたものです.

参考文献

[1] 田村直之, 宋剛秀, 番原睦則. 制約最適化問題と SAT 符号化. 人工知能学会誌, vol25, No1(2010), pp.77-85, 2010.

[2] Carsten Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints,Constraints. In Proc. of CP'05, pp.827–831, 2005.

[3] Carla P. Gomes et al. Near-Uniform Sampling of Combinatorial Spaces Using XOR Constraints. In Proc. of NIPS, pp. 670–676, 2006.

[4] Supratik Chakraborty et al. A Scalable and Nearly Uniform Generator of SAT Witnesses. In Proc. of CAV, pp. 220-216, 2013.

[5] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT Solvers to Cryptographic Problems. SAT, vol. 5584, pp. 244–257, 2009.

[6] Mate Soos and Kuldeep S. Meel. BIRD: Engineering an Efficient CNF-XOR SAT Solver and its Applications to Approximate Model Counting. In AAAI,vol. 33, pp. 1592-1599. 2019.