SATの近似符号化における解網羅度の可変化

財前 昂平　　(指導教員　西村 俊二)

令和 6 年 1 月 23 日

Making Solution Coverage Variable in Approximate Encoding of SAT

Kohei Zaizen　(Academic Advisor　Shunji Nishimura)

概要：SAT とは与えられた論理式を真にする真偽値割り当てが存在するかという問題であり，現実問題を SAT として表現することを SAT 符号化という．符号化された問題は様々な制約で構成され，At-Most-K 制約は頻繁に現れる制約の一つである．この制約を符号化するとき命題変数の増加に伴い計算量が指数関数的に増加する．計算量の増加を抑えるため，近似符号化が提案された．近似符号化は，At-Most-K 制約を正確に表現しない代わりに，与えられた命題変数を 2 等分するなどの分割を行い計算量を大幅に抑える．よってソルバの実行時間が短縮されるが，解は At-Most-K 制約を正確に表現していないため，本来の解を見逃すというリスクを持つ．近似符号化を利用し，現実問題を解決するには，本来の解を見逃すリスクを最小限に抑えつつ，ソルバの実行が終了するように計算量を調節する必要がある．本研究では，近似符号化での分割とは別の分割を行い，両者を併用することで得られる解をどの程度包括できたかを表す解網羅度を可変化させる．可変化させると計算量も変化するため，ソルバの実行が終了するように計算量を調節できる．また，分割パターンの増加による解網羅度の変化を調査した．その結果，網羅度の調節に成功したが，分割パターンを 3 つ以上併用した場合では正確な At-Most-K 制約を表した他の符号化と計算量が変わらなかった．

キーワード：SAT，At-Most-K 制約

1. 緒言

充足可能性問題 (satisfiability problem：SAT) は NP 完全性が証明された最初の問題である．近年，ハードウェアの進歩によって SAT を解くプログラムである SAT ソルバー（ソルバ）が飛躍的に進化し，SAT が推論の基盤技術として注目されている [1]．ソルバは日常生活で遭遇する様々な問題にも有効であり，具体的には，スケジューリング問題，プランニング，制約充足問題，ソフトウェア検証などが挙げられる [1].

問題にソルバを適用させるためには入力を論理式で構成する必要がある．問題を論理式に置き換えることを SAT 符号化といい，符号化された問題は様々な制約で構成される．符号化された問題に頻繁に現れる制約のひとつに At-Most-K 制約がある．At-Most-K 制約とは，真となる命題変数を高々 K 個までとする制約である．At-Most-K 制約を符号化するとき，命題変数の増加に伴って計算量が指数関数的に増加する．計算量の増加を緩和するため，バイナリ符号化（Binary Encoding），逐次カウンタ符号化（The Sequential Counter Encoding），コマンダー符号化（The Commander Encoding）などが提案されている [2]．これらの符号化は正確に At-Most-K 制約を表現するため，符号化する問題の規模が大きくなるにつれて計算量も膨大になる．

一方，近似符号化は，At-Most-K 制約を正確に表現しない代わりに，計算量を大幅に抑えることができる [3]．近似符号化を用いることでソルバの実行時間が短縮できるが，解が元の問題を正確に表現できていないため，本来あるはずの解を見逃す場合がある．ユーザが近似符号化を利用する際には，本来の解を見逃すリスクを最小限に抑えつつ，ソルバの実行が終了するように調節する必要がある．

本研究では近似符号化における解の網羅度の可変化を提案する．図 1 に網羅度を可変化した近似符号化の利用を想定したフローチャートを示す．ユーザは At-Most-K 制約を符号化し，ソルバの実行が終了しない場合に計算量が抑えられた近似符号化を利用する．近似符号化で解が得られない場合，解を見逃した可能性があるため，網羅度を変更し，ソルバの実行が終了するように調節する．本来の解を見逃すリスクを最小限に抑えられ，ソルバの実行が終了できる．よって，問題解決に対するソルバの利便性の向上に寄与できるはずである．

2. 関連研究

2.1 SAT

SAT は，与えられた命題論理式（論理式）を充足する命題変数について真偽値割り当てが存在するかどうかを判定する問題である．真または偽の 2 値をとる命題変数をブール変数といい，ブール変数に選言，連言，否定，含意などの演算を施したものを論理式をいう．SAT にお

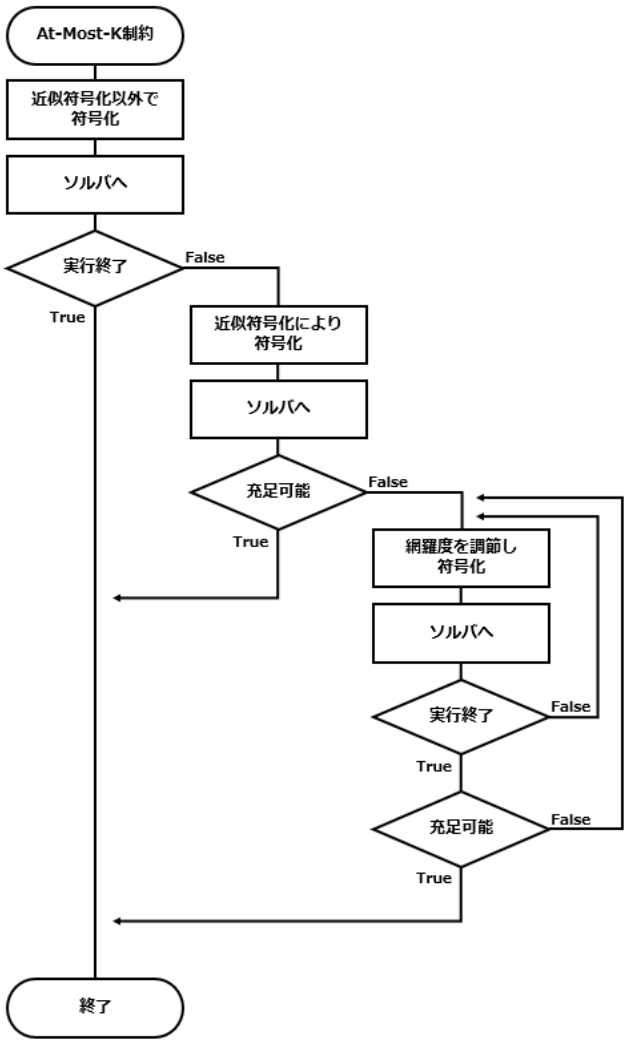

図 1 利用フロー

いて，論理式を真にする真偽値割り当てが存在するとき，その論理式を φ とおくと，φ は充足可能であるという．そのときの割り当ては φ を充足するといい，φ の解となる．論理式を真にする真偽値割り当てが存在しないとき，その論理式は充足不可能であるという．

ソルバに与えられる論理式は，連言標準形である．つまり，選言節（節）を連言で繋いだものであり，各節は命題変数かその否定を表すリテラルの選言である．

SAT を解くプログラムであるソルバの基盤となっているアルゴリズムは 1962 年に Davis らにより開発された DPLL である [4]．DPLL はソルバの基盤として長年使われており，現在の高速ソルバでも使われている．DPLL のアルゴリズムを以下に示す．

Step1. 充足可能もしくは充足不可能となるまで Step2 から Step4 を繰り返す．

Step2. 単位伝播により与えられた論理式の簡単化を行う．

Step3. 命題変数を選択し，真もしくは偽を割り当てる．

Step4. 簡単化した論理式が空であれば充足可能，空節を含めば分岐点まで戻る．

単位伝播とは，1 個のリテラルから構成される節（単位節）に真となるための割り当てを行うことであり，そのリテラルを含む節を除去し，そのリテラルの否定を含む節からはそのリテラルを除去する操作である．例として，3 個の命題変数 x_1, x_2, x_3, x_4 からなる SAT を以下に示す．

$$(\neg x_1 \vee x_2 \vee x_4) \wedge (\neg x_3 \vee x_2) \wedge x_3 \wedge (x_1 \vee \neg x_4) \quad (1)$$

ここで，(1) 式が充足可能かどうかを DPLL アルゴリズムにより判定する．Step2 により，単位節である x_3 に真を割り当てる．そのため，$(\neg x_3 \vee x_2)$ から $\neg x_3$ が除去される．x_3 が除去されたことで x_2 が単位節となり同様に除去される．簡単化された論理式は以下のようになる．

$$(\neg x_1 \vee x_4) \wedge (x_1 \vee \neg x_4) \quad (2)$$

Step3 では，x_1 を選択し真を割り当てる．このとき，$(x_1 \vee \neg x_4)$ は除去され，$(\neg x_1 \vee x_4)$ から $\neg x_1$ が除去される．x_4 は単位節となり，この論理式が充足可能であるといえる．

2.2 At-Most-K 制約

At-Most-K 制約とは, 任意個の命題変数のうち真となる命題変数が高々 K 個までとする制約である．本論文では，n 個の命題変数のうち真となる命題変数が k 個であるという制約を $AtMost\ k/n$ と表す．また，ある集合のうち真となる命題変数の数が k 個であるという制約は $AtMost\ k$ と表し，命題変数の集合 $N = \{x_1, x_2, x_3, ..., x_n\}(n \in \mathbb{N})$ のうち k 個が真である制約は $AtMost\ k/N$ として表す．例として，命題変数が 3 個，真となる命題変数が高々 1 個である $AtMost1/\{x_1, x_2, x_3\}$ の二項符号化による符号化を以下に示す [2].

$$(\neg x_1 \vee \neg x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_3 \vee \neg x_1) \quad (3)$$

(3) 式において，真となる命題変数が 2 個あった場合，偽となる節が現れるため, 全体として偽となる．したがって，真となる命題変数が高々 1 個であることが確認された．また，二項符号化は，命題変数 n 個に対して ${}_nC_{K+1}$ 個の選言節を必要とするため，n が増加するほど節数が膨大となる [2].

2.3 順序符号化

順序符号化は，Crawford と Baker によりジョブショップスケジューリング問題に適用された方法を制約充足問題に適用できるよう一般化されたものである [5].

各整数変数 x について，その整数変数の値が $\{\alpha_1, \alpha_2, \alpha_3, ..., \alpha_n\}(\alpha_1 < \alpha_2 < \alpha_3 < ... < \alpha_n)$ に対

して $x \leqq \alpha_i$ を表す命題変数を用いる．命題変数は $p(x \leqq \alpha_1), p(x \leqq \alpha_2), p(x \leqq \alpha_3), ..., p(x \leqq \alpha_{n-1})$ となり，$p(x \leqq \alpha_n)$ は常に真であるため不要である．命題変数が $x \leqq \alpha_i$ かつその時のみ真となるように以下のように表す．

$$\neg p(x \leqq \alpha_i) \vee p(x \leqq \alpha_{i+1}) \quad (1 \leqq i \leqq n-2) \quad (4)$$

例として，$x \in \{1,2,3,4\}$ の場合，次の 3 個の命題変数 $p(x \leqq 1), p(x \leqq 2), p(x \leqq 3)$ を用い，以下のように表す．

$$\begin{aligned}&(\neg p(x \leqq 1) \vee p(x \leqq 2)) \wedge \\ &(\neg p(x \leqq 2) \vee p(x \leqq 3))\end{aligned} \quad (5)$$

制約は違反する点ではなく違反する領域を列挙する．つまり，範囲 $\{\alpha_1 < x_1 \leqq \beta_1, \alpha_n < x_n \leqq \beta_n\}$ のすべての点 $(x_1, ..., x_n)$ に違反するとき，以下の節を追加する．

$$\begin{aligned}&(p(x_1 \leqq \alpha_1)) \vee \neg p(x_1 \leqq \beta_1) \vee ... \\ &\vee p(x_n \leqq \alpha_n)) \neg p(x_n \leqq \beta_n))\end{aligned} \quad (6)$$

例として，$x, y \in \{1,2,3,4\}$，制約 $x + y \leqq 4$ である場合，まず，各整数変数について命題変数が $x \leqq \alpha_i$ かつその時のみ真となるように (5) 式のように表される．次に，制約は以下のように表される．

$$\begin{aligned}&p(y \leqq 3) \wedge \\ &(p(x \leqq 2) \vee p(y \leqq 2)) \wedge \\ &(p(x \leqq 3) \vee p(y \leqq 1)) \wedge \\ &p(x \leqq 3)\end{aligned} \quad (7)$$

このとき $x \geqq 4$ であるとすると，$p(x \leqq 3)$ が偽であると表される．したがって，(7) 式により全体として偽となるので制約は正しい．

2.4 近似符号化

近似符号化では与えられた命題変数 (入力変数) を分割し，その数に応じて補助変数を追加する [3]．追加された補助変数の一部のみに二項符号化を行い，補助変数の真の数によって入力変数を制約する．近似符号化の基本的な考え方である $2 * 2$ *models* を図 2 に示す．図 2 は木構造を表しており，各円は命題変数，それをまとめた F や $G1$ などを集合とする．木構造であるため，ノード間の階層関係において，上位に位置するノードを親ノード，下位に位置するノードを子ノードと呼び，最上位に位置するノードをルートノードと呼ぶ．ここで，集合 F の左半分である f_1 とその子ノードである集合 $G1$ を例として挙げる．f_1 の真の数によって子ノードである $G1$ を制約する．具体的には以下のように制約される．

- f_1 の真の数が 0 個のとき，$G1$ は $AtMost\ 0/G1$ となる
- f_1 の真の数が 1 個のとき，$G1$ は $AtMost\ 1/G1$ となる

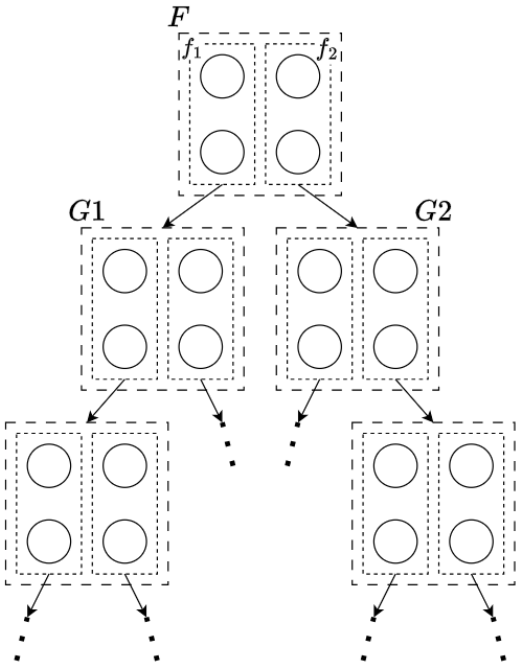

図 2 $2 * 2$ *models*

- f_1 の真の数が 2 個のとき，$G1$ は $AtMost\ 4/G1$ となる

f_1 の真の数が 2 個のとき，$AtMost\ 4/g_1$ となるが，制約として意味を持たないので実際には記述されない．その他の階層関係においても同様に制約される．よって，親ノードの 2 個の命題変数の真の数が n $(0 \leqq n \leqq 2)$ のとき，子ノードは $AtMost\ 2n$ で制約される．ルートノードの 4 個の命題変数に $AtMost\ k$ $(0 \leqq k \leqq 2)$ を与えるのみで $AtMost\ ((k/4) * (2^{m+1}))/2^{m+1}$ の近似的な符号化となる (m は木の高さ)．ルートノードへの At-Most-K 制約は，命題変数の数がわずかであるため，二項符号化により符号化する．例として，$AtMost\ 4/8$ を $2 * 2$ *models* を用いた近似符号化によって符号化する．$AtMost\ ((k/4) * (2^{m+1}))/2^{m+1}$ により，木の高さ $m = 2$，$k = 2$ となり，図 3 に示す．図 3 の集合 G は入力変数であり，2 分割した際の左半分を g_1，右半分を g_2 とする．集合 F は補助変数であり，g_1 と g_2 を子ノードとして持つ．集合 F の左半分である f_1 において，真の数が 1 個であることを P_1，真の数が 2 個であることを P_2 とおき，右半分である f_2 においては，Q_1, Q_2 とおく．これらは順序符号化により符号化され，以下の論理式が得られる．

$$\neg P_2 \vee P_1 \quad (8)$$

$$\neg Q_2 \vee Q_1 \quad (9)$$

また，子ノード g_1 は親ノードの f_1 の真の数により制約され，以下の論理式を得る．

$$\neg P_1 \rightarrow AtMost0/g_1 \quad (10)$$

$$\neg P_2 \rightarrow AtMost2/g_1 \quad (11)$$

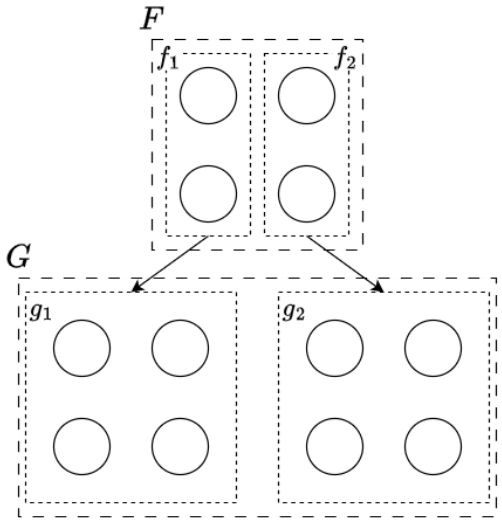

図 3 *AtMost* 4/8

子ノード g_2 も同様に親ノードの f_2 により制約され，以下の論理式を得る．

$$\neg Q_1 \rightarrow AtMost0/g_2 \tag{12}$$

$$\neg Q_2 \rightarrow AtMost2/g_2 \tag{13}$$

$k = 2$ であるため，ルートノードに対し $AtMost$ 2 を与えられ，二項符号化により以下の論理式を得る．

$$\begin{aligned}(\neg P_1 \vee \neg P_2 \vee \neg Q_1) \wedge (\neg P_1 \vee \neg P_2 \vee \neg Q_2) \wedge \\ (\neg P_1 \vee \neg Q_1 \vee \neg Q_2) \wedge (\neg P_2 \vee \neg Q_1 \vee \neg Q_2)\end{aligned} \tag{14}$$

以上により，$AtMost$ 4/8 は近似符号化によって符号化された．しかし，真の数が g_1 で 1 個，g_2 で 3 個となるような解は $AtMost$ 4/8 を満たすが，近似符号化ではこの解を得られない．真の数が g_1 で 1 個，g_2 で 3 個となる解を得られるとすると，f_1 で真が 1 個，f_2 で真が 2 個必要となり，P_1 が真，Q_2 が真となる．この場合，(9) 式から Q_1 も真となる．このとき，(14) 式の $(\neg P_1 \vee \neg Q_1 \vee \neg Q_2)$ の部分が偽となり，全体として偽となる．したがって，g_1 で 1 個，g_2 で 3 個となる解は得られない．同様に，真の数が g_1 で 3 個，g_2 で 1 個となる場合も $AtMost$ 4/8 を満たすが，近似符号化では解を得られない．このことから近似符号化は At-Most-K 制約を正確に表現できない．一方で，生成される式を抑えたためソルバの実行時間の短縮ができる．

3. 提案手法

2.4 節に示すように近似符号化は 8 個の入力変数を与えられた場合，4 個ずつに分割し，符号化を行う．具体的には入力変数を $\{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$ としたとき，$\{x_1, x_2, x_3, x_4\}, \{x_5, x_6, x_7, x_8\}$ と分割する．しかし，このときの分割のパターンはこの他にも $\{x_1, x_2, x_5, x_6\}, \{x_3, x_4, x_7, x_8\}$ などがあり，そのパターン数は，合計で 35 パターン存在する．近似符号化は 1 パターンのみで符号化を行うため，得られない解が存在し，本来の解を見逃すリスクがある．本研究では，このパターン数を調節することで解網羅度を可変化し，本来の解を見逃すリスクを最小限に抑える．例として，$AtMost$ 4/8 に対し 2 パターンを併用した場合と 3 パターンを併用した場合をそれぞれ図 4，図 5 に示す．木構造や集合，命題変数に関しては 2.4 節と同様の設定とする．図 4 は，分割を左右で行った場合と上下で行った場合として 2 パターン用い，図 5 は，図 4 の分割に集合 I のような分割を加える．

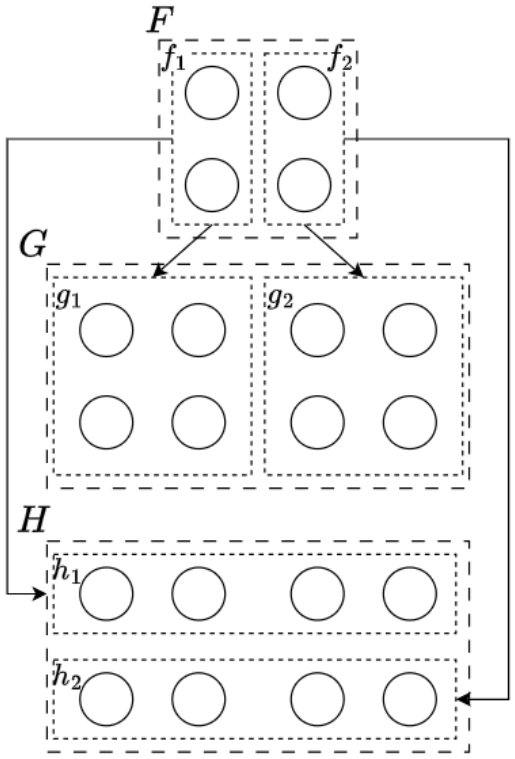

図 4 2 パターン

3.1 2 パターン

図 4 において，集合 G と集合 H は同じ集合であるが視認性を良くするため表記を分けている．2.4 節同様に符号化されるが，親ノードから子ノードへの制約は変化する．集合 G での分割パターンを適用した際の親ノードから子ノードへの制約を以下に示す．

$$\begin{aligned}\neg P_1 &\rightarrow AtMost0/g_1 \\ \neg P_2 &\rightarrow AtMost2/g_1 \\ \neg Q_1 &\rightarrow AtMost0/g_2 \\ \neg Q_2 &\rightarrow AtMost2/g_2\end{aligned} \tag{15}$$

同様に，集合 H での分割パターンを適用した際の親ノードから子ノードへの制約を以下に示す．

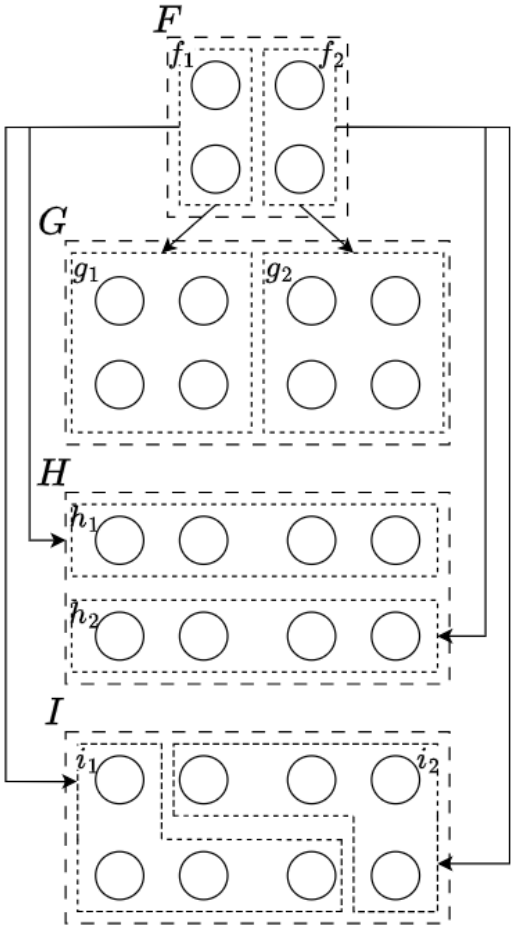

図 5 3 パターン

$$\begin{aligned}
\neg P_1 &\rightarrow AtMost0/h_1 \\
\neg P_2 &\rightarrow AtMost2/h_1 \\
\neg Q_1 &\rightarrow AtMost0/h_2 \\
\neg Q_2 &\rightarrow AtMost2/h_2
\end{aligned} \tag{16}$$

(15) 式と (16) 式を選言で繋ぐことで，2 パターンを併用したときの解を得る．ここで，ソルバの入力は連言標準形であり，(15) 式と (16) 式を選言で繋ぐだけでは連言標準形に変換する必要がある．しかし，変換すると節数が増加し，計算量が増える．そのため，スイッチの役割を持つ補助変数を 1 つ追加することで節数の増加を抑制する．この補助変数を s とおくと，(15) 式と (16) 式は以下のようになる．

$$\begin{aligned}
s \wedge \neg P_1 &\rightarrow AtMost0/g_1 \\
s \wedge \neg P_2 &\rightarrow AtMost2/g_1 \\
s \wedge \neg Q_1 &\rightarrow AtMost0/g_2 \\
s \wedge \neg Q_2 &\rightarrow AtMost2/g_2
\end{aligned} \tag{17}$$

$$\begin{aligned}
\neg s \wedge \neg P_1 &\rightarrow AtMost0/h_1 \\
\neg s \wedge \neg P_2 &\rightarrow AtMost2/h_1 \\
\neg s \wedge \neg Q_1 &\rightarrow AtMost0/h_2 \\
\neg s \wedge \neg Q_2 &\rightarrow AtMost2/h_2
\end{aligned} \tag{18}$$

s が真となるとき集合 G での分割パターンが適用され，偽であるとき，集合 H での分割パターンが適用されるため，(15) 式と (16) 式を連言で繋ぐことができる．以上により，$AtMost$ 4/8 は 2 パターンでの近似符号化によって符号化された．

2 パターンを併用することで 2.4 節では得られなった真の数が g_1 で 1 個，g_2 で 3 個となるような解を得る．具体的には図 4 において，$x_1 = x_2 = x_4 = x_7 = True, x_3 = x_5 = x_6 = x_8 = False$ であった場合，集合 G での分割パターンにおいて，解は得られないが，集合 H での分割パターンでは解を得られる．しかし，$x_1 = x_2 = x_4 = x_5 = True, x_3 = x_6 = x_7 = x_8 = False$ であった場合など，どちらの分割パターンでも得られない解も存在する．

3.2 3 パターン

図 5 において，2 パターンでの符号化と同様に集合 G と集合 H と集合 I は同じ集合であるが，視認性を良くするため表記を分けている．2 パターンでの符号化と同様にスイッチの役割を持つ補助変数を追加する．3 パターンを切り替えなければならないため，補助変数の数は 3 つとなる．この 3 つの補助変数を s_1, s_2, s_3 とおくと，f_1, f_2 子ノード $\{g_1, g_2\}, \{h_1, h_2\}, \{i_1, i_2\}$ への制約は以下のようになる．

$$\begin{aligned}
s_1 \wedge \neg P_1 &\rightarrow AtMost0/g_1 \\
s_1 \wedge \neg P_2 &\rightarrow AtMost2/g_1 \\
s_1 \wedge \neg Q_1 &\rightarrow AtMost0/g_2 \\
s_1 \wedge \neg Q_2 &\rightarrow AtMost2/g_2
\end{aligned} \tag{19}$$

$$\begin{aligned}
s_2 \wedge \neg P_1 &\rightarrow AtMost0/h_1 \\
s_2 \wedge \neg P_2 &\rightarrow AtMost2/h_1 \\
s_2 \wedge \neg Q_1 &\rightarrow AtMost0/h_2 \\
s_2 \wedge \neg Q_2 &\rightarrow AtMost2/h_2
\end{aligned} \tag{20}$$

$$\begin{aligned}
s_3 \wedge \neg P_1 &\rightarrow AtMost0/i_1 \\
s_3 \wedge \neg P_2 &\rightarrow AtMost2/i_1 \\
s_3 \wedge \neg Q_1 &\rightarrow AtMost0/i_2 \\
s_3 \wedge \neg Q_2 &\rightarrow AtMost2/i_2
\end{aligned} \tag{21}$$

次に，3 つの補助変数のうち，ただ 1 つが真となる制約を加える．

$$
\begin{aligned}
&(\neg s_1 \vee \neg s_2) \wedge \\
&(\neg s_1 \vee \neg s_3) \wedge \\
&(\neg s_2 \vee \neg s_3) \wedge \\
&(s_1 \vee s_2 \vee s_3)
\end{aligned} \tag{22}
$$

以上により，$AtMost$ 4/8 は 3 パターンでの近似符号化によって符号化された．

3 パターンを併用することで，2 パターンの場合では得られなった解を得られる．図 5 において，$x_1 = x_2 = x_4 = x_5 = True, x_3 = x_6 = x_7 = x_8 = False$ となる解は，集合 G と集合 H での分割パターンでは得られないが，集合 I での分割パターンでは得られる．

得られる解を増加させるためには，パターン数を増やすことが有効である．しかし，新たな節の追加により計算量も増加するため，計算資源に応じてパターン数を調節するといった利用が望まれる．

4. 実験

本節ではパターン数を増加させたときの解網羅度と論理式の量を調べ，解網羅度を可変化できたかを調べる．

4.1 実験方法

本実験では，$AtMost$ 4/8 について，入力変数を $\{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$ とおき，符号化を行う．符号化法として二項符号化，近似符号化，2 パターンでの近似符号化，3 パターンでの近似符号化の 4 つの方法を用いる．解網羅度の可変化を確認するため，リテラル数，節数，解の数 (solution count)，解網羅度 (coverage)，計算時間を求める．近似符号化には $2 * 2$ $models$ を適用し，全 35 パターンを表 1 に示す．$AtMost$ 4/8 を正確に表現した際に得られる解の数は 163 個であり，解網羅度は以下の計算により求めることとする．

$$
coverage = (solution\ count)/163 \tag{23}
$$

また，解を数える際には，生成した論理式の命題変数に考えられうる全ての真理値を割り当てることで求め，そのときに要した時間を計算時間とする．

2 パターンでの近似符号化について，1 つ目のパターンは表 1 の pattern1 に固定し，2 つ目は pattern2 から pattern35 までの 34 通りで解の数の変化を調べる．

3 パターンでの近似符号化について，1 つ目のパターンは 2 パターンでの近似符号化同様，表 1 の pattern1 に固定し，2 つ目は pattern1 のそれぞれから 1 つずつ入れ替えた pattern2 と pattern1 のそれぞれから 2 つずつ入れ替えた pattern10 とし，3 つ目は 2 つ目が pattern2 であった場合は pattern2 のそれぞれから 1 つずつ入れ替えた pattern10 と 2 つずつ入れ替えた pattern18,pattern24 とし，2 つ目が pattern10 であった場合は pattern10 のそれぞれから 1 つずつ入れ替えた pattern18,pattern31 と 2 つずつ入れ替えた pattern15 とし，解の数の変化を調べる．

表 1 パターン一覧

パターン	組み合わせ
pattern1	$\{x_1, x_2, x_3, x_4\}, \{x_5, x_6, x_7, x_8\}$
pattern2	$\{x_1, x_2, x_3, x_5\}, \{x_4, x_6, x_7, x_8\}$
pattern3	$\{x_1, x_2, x_3, x_6\}, \{x_4, x_5, x_7, x_8\}$
pattern4	$\{x_1, x_2, x_3, x_7\}, \{x_4, x_5, x_6, x_8\}$
pattern5	$\{x_1, x_2, x_3, x_8\}, \{x_4, x_5, x_6, x_7\}$
pattern6	$\{x_1, x_2, x_4, x_5\}, \{x_3, x_6, x_7, x_8\}$
pattern7	$\{x_1, x_2, x_4, x_6\}, \{x_3, x_5, x_7, x_8\}$
pattern8	$\{x_1, x_2, x_4, x_7\}, \{x_3, x_5, x_6, x_8\}$
pattern9	$\{x_1, x_2, x_4, x_8\}, \{x_3, x_5, x_6, x_7\}$
pattern10	$\{x_1, x_2, x_5, x_6\}, \{x_3, x_4, x_7, x_8\}$
pattern11	$\{x_1, x_2, x_5, x_7\}, \{x_3, x_4, x_6, x_8\}$
pattern12	$\{x_1, x_2, x_5, x_8\}, \{x_3, x_4, x_6, x_7\}$
pattern13	$\{x_1, x_2, x_6, x_7\}, \{x_3, x_4, x_5, x_8\}$
pattern14	$\{x_1, x_2, x_6, x_8\}, \{x_3, x_4, x_5, x_7\}$
pattern15	$\{x_1, x_2, x_7, x_8\}, \{x_3, x_4, x_5, x_6\}$
pattern16	$\{x_1, x_3, x_4, x_5\}, \{x_2, x_6, x_7, x_8\}$
pattern17	$\{x_1, x_3, x_4, x_6\}, \{x_2, x_5, x_7, x_8\}$
pattern18	$\{x_1, x_3, x_4, x_7\}, \{x_2, x_5, x_6, x_8\}$
pattern19	$\{x_1, x_3, x_4, x_8\}, \{x_2, x_5, x_6, x_7\}$
pattern20	$\{x_1, x_3, x_5, x_6\}, \{x_2, x_4, x_7, x_8\}$
pattern21	$\{x_1, x_3, x_5, x_7\}, \{x_2, x_4, x_6, x_8\}$
pattern22	$\{x_1, x_3, x_5, x_8\}, \{x_2, x_4, x_6, x_7\}$
pattern23	$\{x_1, x_3, x_6, x_7\}, \{x_2, x_4, x_5, x_8\}$
pattern24	$\{x_1, x_3, x_6, x_8\}, \{x_2, x_4, x_5, x_7\}$
pattern25	$\{x_1, x_3, x_7, x_8\}, \{x_2, x_4, x_5, x_6\}$
pattern26	$\{x_1, x_4, x_5, x_6\}, \{x_2, x_3, x_7, x_8\}$
pattern27	$\{x_1, x_4, x_5, x_7\}, \{x_2, x_3, x_6, x_8\}$
pattern28	$\{x_1, x_4, x_5, x_8\}, \{x_2, x_3, x_6, x_7\}$
pattern29	$\{x_1, x_4, x_6, x_7\}, \{x_2, x_3, x_5, x_8\}$
pattern30	$\{x_1, x_4, x_6, x_8\}, \{x_2, x_3, x_5, x_7\}$
pattern31	$\{x_1, x_4, x_7, x_8\}, \{x_2, x_3, x_5, x_6\}$
pattern32	$\{x_1, x_5, x_6, x_7\}, \{x_2, x_3, x_4, x_8\}$
pattern33	$\{x_1, x_5, x_6, x_8\}, \{x_2, x_3, x_4, x_7\}$
pattern34	$\{x_1, x_5, x_7, x_8\}, \{x_2, x_3, x_4, x_6\}$
pattern35	$\{x_1, x_6, x_7, x_8\}, \{x_2, x_3, x_4, x_5\}$

4.2 実験結果

4.1 により行った結果を表 2 に示す．表 2 において，2 パターンでの近似符号化で 1 つ目のパターンが pattern1，2 つ目のパターンが pattern2 である場合，「1,2」と表記する．同様に 3 パターンでの近似符号化は「1,2,3」と表記している．また，coverage の値と計算時間は有効数字を 4 桁としている．表 2 より，2 パターンでの近似符号化は最大で 151 個の解を得られ，解網羅度は 0.9264 である．また，3 パターンでの近似符号化は最大で 163 個の解を得られ，網羅ができた．よって，パターン数を増やすことで解網羅度の可変化に成功した．

表 2 実験結果

	リテラル数	節数	解の数	coverage	時間 (ms)
二項	280	56	163	1.000	2087
近似	64	22	131	0.8037	36.55
1,2	144	38	151	0.9264	109.4
1,3	144	38	151	0.9264	101.0
1,4	144	38	151	0.9264	96.55
1,5	144	38	151	0.9264	107.3
1,6	144	38	151	0.9264	103.3
1,7	144	38	151	0.9264	98.66
1,8	144	38	151	0.9264	99.04
1,9	144	38	151	0.9264	105.5
1,10	144	38	147	0.9018	97.74
1,11	144	38	147	0.9018	107.2
1,12	144	38	147	0.9018	102.6
1,13	144	38	147	0.9018	96.40
1,14	144	38	147	0.9018	96.79
1,15	144	38	147	0.9018	102.2
1,16	144	38	151	0.9264	102.4
1,17	144	38	151	0.9264	99.73
1,18	144	38	151	0.9264	99.79
1,19	144	38	151	0.9264	102.6
1,20	144	38	147	0.9018	96.42
1,21	144	38	147	0.9018	96.79
1,22	144	38	147	0.9018	96.07
1,23	144	38	147	0.9018	108.0
1,24	144	38	147	0.9018	97.52
1,25	144	38	147	0.9018	98.97
1,26	144	38	147	0.9018	96.63
1,27	144	38	147	0.9018	101.6
1,28	144	38	147	0.9018	96.13
1,29	144	38	147	0.9018	99.00
1,30	144	38	147	0.9018	104.8
1,31	144	38	147	0.9018	97.23
1,32	144	38	151	0.9264	96.52
1,33	144	38	151	0.9264	99.87
1,34	144	38	151	0.9264	96.84
1,35	144	38	151	0.9264	100.1
1,2,10	217	58	159	0.9755	559.3
1,2,18	217	58	159	0.9755	540.0
1,2,24	217	58	155	0.9509	578.6
1,10,18	217	58	159	0.9755	524.7
1,10,31	217	58	155	0.9509	569.5
1,10,15	217	58	163	1.000	539.7

5. 考察

第一に，リテラル数について考察する．表 2 の実験結果より，近似符号化では 64 であったリテラル数が 2 パターンでの近似符号化では 144 であり，約 2.3 倍となっている．これはパターン数を 2 倍にしたことによって，親ノードから子ノードへの制約が増え，その制約にスイッチの役割を行う補助変数が追加されたこととが要因だと考えられる．3 パターンでの近似符号化でも 217 であり，近似符号化でのリテラル数の約 3.4 倍となっている．これは 2 パターンでの近似符号化と同様の理由だと考えられる．さらに補助変数の数が 2 パターンでの近似符号化よりも増えたこともリテラル数の増加につながる要因だと考えられる．このようにパターン数が増加するとリテラル数も増加するため，分割パターンを 4 つ併用させた場合，二項符号化のリテラル数より劣ると考えられる．そのため，リテラル数の観点から，分割パターンを 4 つ以上併用させる場合は二項符号化を利用するほうが優位であると考えられる．

第二に，節数について考察する．表 2 の実験結果より，近似符号化では 22 であった節数が 2 パターンでの近似符号化では 38 であり，約 1.7 倍となっている．リテラル数と違い，節数は 2 倍以上にならない．これはパターン数を 2 倍にしたことにより親ノードから子ノードへの制約は増えるが，補助変数の影響を受けないため，リテラル数のように 2 倍以上とならないと考えられる．3 パターンでの近似符号化でも同様の理由で節数が 3 倍以上にならないと考えられる．また，3 の (22) 式のように 3 パターンでの近似符号化ではスイッチの役割を持つ補助変数に対し分割パターンの切り替えを行う制約を加える必要がある．この制約について順序符号化を利用することで節数の短縮が行えると考えられる．3 パターンでの近似符号化の節数は二項符号化の節数よりも多く，分割パターンを 3 つ以上併用させる場合は二項符号化を利用するほうが優位であると考えられる．

第三に，解の数と解網羅度 (coverage) について考察する．表 2 の実験結果より，2 パターンでの近似符号化で解の数が 147 と 151 と異なっている．解の数が 147 である箇所は pattern1 のそれぞれから 2 つずつ入れ替えた分割パターンであり，151 である箇所は pattern1 のそれぞれから 1 つずつ入れ替えた分割パターンである．2 つ目の分割パターンが pattern2 であるときと，pattern21 であるときを例として取り上げる．近似符号化では入力変数を分割したとき，半分の命題変数がすべて真となる解を得られる．pattern2 では $x_1 = x_2 = x_3 = x_5 = True, x_4 = x_6 = x_7 = x_8 = False$ のような解であり，pattern21 では $x_1 = x_3 = x_5 = x_7 = True, x_2 = x_4 = x_6 = x_8 = False$ のような解である．このとき，pattern21 の解は 1 つ目の分割パターンである pattern1 で得られる解でもある．故に，1 つ目の分割パターンのそれぞれから 2 つずつ入れ替えた分割パターンを併用した場合，得られる解が減少すると考えられる．3 パターンでの近似符号化で，併用する分割パターンの組み合わせを変えた場合に解の数が異なっている要因も同様だと考えられる．しかし，1 つ目の分割パターンのそれぞれから 2 つずつ入れ替えた分割パターンを併用している pattern1，pattern10，pattern15 の組み合わせでは網羅することができているため，絶対的な

要因とは言えない．また，今回の実験では $AtMost$ 4/8 に限定しため，3 パターンでの近似符号化により網羅できるという結果を得られた．だが，$AtMost$ 8/16 など入力変数の数が変化した場合は，3 パターンでの近似符号化で網羅できないと考える．

第四に，論理式の量と解網羅度について考察する．2 パターンでの近似符号化の論理式の量は，既存の近似符号化の論理式の量より約 2 倍増加したが，解網羅度は 10% 程度の増加である．このことからわずかな論理式の量で解を多く得られるのは近似符号化である．しかし，2 パターンでの近似符号化の論理式の量をソルバが解くことができるならば，既存の近似符号化より利用価値が高いと考えられる．

第五に，計算時間について考察する．近似符号化での計算時間は 36.55 ミリ秒，2 パターンでの近似符号化の計算時間は約 100 ミリ秒であり，約 2.7 倍である．3 パターンでの近似符号化の計算時間は約 550 ミリ秒であり，近似符号化での計算時間の約 15 倍である．パターン数が増加すると，計算時間は大きく増加する．しかし，これらの符号化は，二項符号化に比べ計算時間が大幅に抑えられているため，二項符号化より実用的であると考えられる．

最後に，符号化法について考察する．3 パターンでの近似符号化は二項符号化と比べ，リテラル数や節数に関して大きな差がないにもかかわらず解網羅度に関しては劣る．そのため，パターン数を 3 つ以上にする場合については他の符号化を利用すべきと考える．2 パターンでの近似符号化は二項符号化に比べ，リテラル数や節数，計算時間が抑えられ，近似符号化より解網羅度が上昇していることから本来の解を見逃すリスクを抑えつつ，ソルバの実行が終了するよう調節可能だと考えられる．

6. 今後の課題

近似符号化において，パターン数を増やすことで解網羅度の可変化に成功したが，実用化のためには，他の符号化と比較し評価する必要がある．また，$AtMost$ 4/8 に限定したため，入力変数が増加した際のリテラル数や節数，計算時間に関しても評価しなければならない．分割パターンを 3 つ以上併用させる場合は，リテラル数や節数の増加が目立った．スイッチの役割を持つ補助変数に関する制約を直接論理式にせず，他の符号化を利用し，リテラル数や節数の増加を抑えるべきである．

7. 結言

近似符号化において本来の解を見逃すリスクを最小限に抑え，ソルバの実行が終了するように調節を行う必要があった．本研究では，近似符号化を構成するにあたって行われる分割のパターンを複数併用することで解決を試みた．パターン数を増加させることで解網羅度の可変化に成功した．しかし，6 節に示すような課題も残っている．これらの課題を解決したとき，近似符号化はより緻密に解網羅度を可変化する符号化法になると考える．

謝辞

本研究を進めるにあたって，さまざまなご指導を頂きました西村俊二准教授に深く感謝申し上げます．また，多くの知識やご指導をくださいました同研究室の先輩・同期の皆様に厚く御礼申し上げます．

参考文献

[1] 宋 剛秀, 番原 睦則, 田村 直之, 鍋島 英知,「SAT ソルバーの最新動向と利用技術」, コンピュータソフトウェア 35 巻.4 号 (2018)p. 72-92

[2] Alan M,Frisch and Paul A,Giannaros. ,“SAT Encodings of the At-Most-k Constraint Some Old, Some New, Some Fast, Some Slow”,Proc. of the Ninth Int. Workshop of Constraint Modelling and Reformulation , 2010

[3] Nishimura, S. Approximate-At-Most-k Encoding of SAT for Soft Constraints.The 14th Pragmatics of SAT International Workshop (PoS2023), Alghero, Italy, 4–8 July 2023

[4] 井上 克巳, 田村 直之,「SAT ソルバーの基礎 (¡特集¿ 最近の SAT 技術の発展)」, 人工知能学会誌 25 巻 1 号 (2010) p. 57-67

[5] 田村 直之, 丹生 智也, 番原 睦則,「制約最適化問題と SAT 符号化 (¡ 特集¿ 最近の SAT 技術の発展).」人工知能学会誌 25 巻 1 号 (2010) p.77-85.