SATソルバーの使い方 ―問題をSATに符号化する方法―

田村 直之[1,a)]　宋 剛秀[1,b)]　番原 睦則[1,c)]

概要：SAT ソルバーは，与えられた連言標準形の命題論理式 (CNF 式) を満たす解を探索するプログラムである．Knuth による著名な教科書 The Art of Computer Programming の最新分冊では，300 ページ以上もの分量が SAT ソルバーとその応用の説明に割かれ，最近のホットなトピックの一つとなっている．このような SAT ソルバーの発展を背景とし，解きたい問題を CNF 式に変換 (SAT 符号化) し SAT ソルバーで解を求める手法が注目されている．しかし，解きたい問題が複雑な場合，それを直接 SAT 符号化することはかなり面倒な仕事だ．代わりに，まず与えられた問題を整数変数上の制約モデルとして定式化し，次にその制約モデルの SAT 符号化を考える方法が有効である．そこで，本発表では上記の Knuth の教科書でも取り上げられている例題を題材とし，制約モデル・SAT 符号化の選択肢について説明・比較を行い，SAT ソルバーをより有効に利用する方法を紹介する．

キーワード：SAT ソルバー, SAT 符号化, 制約モデル

1. はじめに

2015 年の年末に，チューリング賞受賞者の Knuth による有名な教科書 “The Art of Computer Programming” の最新分冊 [6] が出版された．そこでは，300 ページ以上もの分量が “SAT” に費されている．

SAT とは命題論理の充足可能性 (satisfiability) の略であり，与えられた連言標準形 (Conjunctive Normal Form; CNF) の命題論理式 (CNF 式) を真にする値割り当てが存在するか否かを判定する問題である．近年になって，SAT を解くためのプログラムである **SAT ソルバー** の性能が飛躍的に向上し，ハードウェア検証，ソフトウェア検証，制約充足等の問題について，それらを命題論理の CNF 式に変換 (**SAT 符号化**; SAT encoding) した後，SAT ソルバーに解かせることで元の問題の解を求める方法が実用的になっている．Knuth も，前述の [6] の序文で「SAT は，非常に多くの問題を解くための鍵となることから，明らかに killer app だ」と述べている．

しかし，解きたい問題が複雑な場合，それを直接 SAT 符号化することはかなり面倒な仕事だ．代わりに，まず与えられた問題を整数変数上の制約モデルとして定式化し，次にその制約モデルの SAT 符号化を考える方法が有効である [18]．

そこで，本稿では上記の Knuth の教科書でも取り上げられているクイーングラフ彩色問題を題材とし，制約モデル・SAT 符号化の選択肢について説明・比較を行い，SAT ソルバーをより有効に利

[1] 神戸大学 情報基盤センター
[a)] tamura@kobe-u.ac.jp
[b)] soh@lion.kobe-u.ac.jp
[c)] banbara@kobe-u.ac.jp

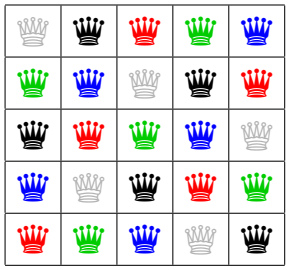

図 1　5 次クイーングラフ彩色問題の 2 重周期解の例

用する方法を紹介する．

なお，SAT ソルバーの概要について記述されている文献には [1], [2], [3], [5], [10] などがある．読者の参考にされたい．

2. クイーングラフ彩色問題

クイーングラフ彩色問題 (Queen graph coloring problem) は，N 個ずつの N 個のグループからなるクイーン (計 N^2 個) を，$N \times N$ のチェス盤に，同じグループのクイーン同士が互いに取られないように配置する問題である．

この問題は N^2 頂点から成るクイーングラフ [*1] を N 色で塗り分ける**グラフ彩色問題** (graph coloring problem) でもある．つまり，一手でクイーンが移動できるマス同士が同色にならないように，各マスを N 色で塗り分ける問題とも見なせる．また N 次 **対角ラテン方陣** (diagonal Latin square) と呼ばれることもある [*2]．

George Pólya は $N \equiv \pm 1 \pmod 6$ の時にだけ 2 重周期的な解が存在することを示した．図 1 に 5 次 ($N = 5$) の (2 重周期的な) 解を示す．1 行目を 2 列分だけ右に回転させたものが 2 行目になっており，以下も同様である．同色に塗られたクイーン同士は互いに取られない配置になっている．

計算機により $N = 2, 3, 4, 6, 8, 9, 10$ の場合に解が存在しないことが確かめられたため，しばらくの間，Pólya の条件はクイーングラフ彩色問題の解の存在を表す必要十分条件と考えられていた．実際，1979 年に Martin Gardner は $N \equiv \pm 1 \pmod 6$ の時にだけ解が存在すると誤解して述べている．しかし，2003 年に $N = 12$ の解が Michel Vasquez と Günter Stertenbrink によって独立に発見された (図 2 参照)．もちろんこの解は 2 重周期的ではない．その後，2005 年までに 11 以上 26 以下のすべての N に対し解が発見されている．

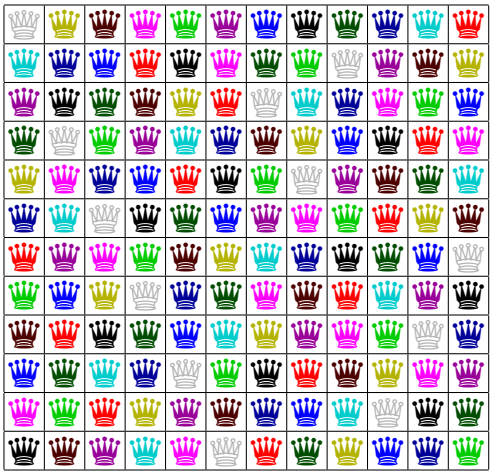

図 2　12 次クイーングラフ彩色問題の解の例

グラフ彩色問題は Knuth の教科書でも最初の部分 (p.6–8) で取り上げられ，さらに 99–100 ページで種々の SAT 符号化を比較するためにクイーングラフ彩色問題が用いられている．

3. クイーングラフ彩色問題の制約モデル

クイーングラフ彩色問題の制約モデルを 2 通り示す．まず，集合 **N**, **U**, **D** を次のように定義する．

$$\mathbf{N} = \{0, 1, \ldots, N-1\}$$
$$\mathbf{U} = \{0, 1, 2, \ldots, 2N-2\}$$
$$\mathbf{D} = \{1-N, 2-N, \ldots, N-1\}$$

N は，行番号 i, 列番号 j, クイーンの色 k の取り得る値の集合を表している．**U** は $i+j$ の取り得る値の集合であり，右上がりの対角線に対応する．同様に **D** は $i-j$ の取り得る値の集合であり，右下がりの対角線に対応する．

[*1] チェス盤のマスを頂点とし，同一行，同一列，同一対角線上にあるマス同士を辺で結んだグラフ．ここでは，主対角線に並行な斜めの線すべてを対角線と呼ぶ．

[*2] 各行と各列に加え，各対角線についても互いに異なる数が現れるラテン方陣．対角ラテン方陣には，別の定義もある．

3.1 色変数モデル

色変数モデル は，クイーンの色を整数変数とした制約モデルである．

位置 (i, j) に配置されたクイーンの色を整数変数 c_{ij} で表す．

$$c_{ij} \in \mathbf{N} \qquad (i, j \in \mathbf{N})$$

各行 i について，配置されている N 個のクイーンの色が互いに異なるという条件は，以下の制約で表される．

$$\mathrm{alldiff}\,\{c_{ij} \mid j \in \mathbf{N}\} \quad (i \in \mathbf{N})$$

ここで $\mathrm{alldiff}\,\{z_1, z_2, \ldots, z_n\}$ は，**alldifferent 制約** と呼ばれるもので z_i の値が互いに異なることを意味する．

同様に，各列，各右上がり対角線，各右下がり対角線に配置されているクイーンの色も互いに異なるから以下の制約が得られる．

$$\begin{aligned}
&\mathrm{alldiff}\,\{c_{ij} \mid i \in \mathbf{N}\} && (j \in \mathbf{N})\\
&\mathrm{alldiff}\,\{c_{ij} \mid i, j \in \mathbf{N},\ i + j = u\} && (u \in \mathbf{U})\\
&\mathrm{alldiff}\,\{c_{ij} \mid i, j \in \mathbf{N},\ i - j = d\} && (d \in \mathbf{D})
\end{aligned}$$

最後に，対称的な解を除去するため，0 行目 j 列目に配置されるクイーンの色を j に固定する．

$$c_{0j} = j \qquad (j \in \mathbf{N})$$

3.2 位置変数モデル

位置変数モデル は，各行に配置されているクイーンの列番号を整数変数とした制約モデルである．

整数変数 y_{ik} の値は，色 k のクイーンが行 i で配置されている列番号を表す．

$$y_{ik} \in \mathbf{N} \qquad (i, k \in \mathbf{N})$$

各行 i に配置されている N 色のクイーンの列番号は互いに異なるという条件は，以下の制約で表される．

$$\mathrm{alldiff}\,\{y_{ik} \mid k \in \mathbf{N}\} \qquad (i \in \mathbf{N})$$

また，各色のクイーンに対し以下の制約が得られる．

$$\begin{aligned}
&\mathrm{alldiff}\,\{y_{ik} \mid i \in \mathbf{N}\} && (k \in \mathbf{N})\\
&\mathrm{alldiff}\,\{y_{ik} + i \mid i \in \mathbf{N}\} && (k \in \mathbf{N})\\
&\mathrm{alldiff}\,\{y_{ik} - i \mid i \in \mathbf{N}\} && (k \in \mathbf{N})
\end{aligned}$$

最初の制約は，同一の列番号にクイーンが配置されないことを表している．次の制約は 列番号 + 行番号 の値が互いに異なること，すなわち各右上がり対角線上に 2 つ以上のクイーンが配置されないことを表す．同様に，最後の制約は 列番号 – 行番号 の値が互いに異なること，すなわち各右下がり対角線上に 2 つ以上のクイーンが配置されないことを表す．

0 行目 j 列目に配置されるクイーンの色を j に固定する制約は以下の通りである．

$$y_{0j} = j \qquad (j \in \mathbf{N})$$

4. クイーングラフ彩色問題の SAT 符号化

次に，前節で記述した制約モデルの SAT 符号化について述べる．

4.1 整数変数の SAT 符号化

色変数モデルおよび位置変数モデルでは，整数変数が利用されている．整数変数の SAT 符号化には様々な方法が提案されているが，ここでは直接符号化 [4] と順序符号化 [16] について説明する．

4.1.1 直接符号化

SAT 符号化したい整数変数を z とし，取り得る値の集合 (**ドメイン** と呼ばれる) を $\{l, l+1, \ldots, u\}$ とする．すなわち $z \in \{l, l+1, \ldots, u\}$ である．

直接符号化 (direct encoding) では，各変数 z と，そのドメインの各値 a に対し，$z = a$ を意味するブール変数 $P(z = a)$ を導入する [4].

$$P(z = a) \in \{0, 1\} \quad (l \le a \le u)$$

この時，整数変数 z は以下の節に符号化できる．

$$\bigvee_{a=l}^{u} P(z = a)$$

$$\neg P(z = a) \vee \neg P(z = b) \quad (l \le a < b \le u)$$

表 1 整数変数の値と順序符号化後のブール変数の値の対応

z	$P(z \geq 1)$	$P(z \geq 2)$	$P(z \geq 3)$
0	0	0	0
1	1	0	0
2	1	1	0
3	1	1	1

最初の節は，整数変数 z が l から u の値を少なくとも 1 つ取ることを表し，次の節は，2 つ以上の値を取らないことを表している．

たとえば $z \in \{0,1,2,3\}$ の場合の節は以下の 7 つである．

$$P(z=0) \vee P(z=1) \vee P(z=2) \vee P(z=3)$$
$$\neg P(z=0) \vee \neg P(z=1)$$
$$\neg P(z=0) \vee \neg P(z=2)$$
$$\neg P(z=0) \vee \neg P(z=3)$$
$$\neg P(z=1) \vee \neg P(z=2)$$
$$\neg P(z=1) \vee \neg P(z=3)$$
$$\neg P(z=2) \vee \neg P(z=3)$$

制約 $z = a$ および $z \neq a$ は，それぞれ単に $P(z=a)$ と $\neg P(z=a)$ で表せる．

4.1.2 順序符号化

順序符号化 (order encoding) では，各変数 z と，最小値以外のドメインの各値 a に対し，$z \geq a$ を意味するブール変数 $P(z \geq a)$ を導入する [16].

$$P(z \geq a) \in \{0,1\} \quad (l < a \leq u)$$

z の SAT 符号化として $z \geq a \Rightarrow z \geq a-1$ を意味する節を加える．

$$P(z \geq a-1) \vee \neg P(z \geq a) \quad (l < a \leq u)$$

例えば $z \in \{0,1,2,3\}$ の場合，以下の 2 節になる．

$$P(z \geq 1) \vee \neg P(z \geq 2)$$
$$P(z \geq 2) \vee \neg P(z \geq 3)$$

表 1 に $z \in \{0,1,2,3\}$ の場合について，z の値と，順序符号化した節を充足する $P(z \geq a)$ の値の対応を示す．

制約 $z = a$ および $z \neq a$ は，それぞれ $P(z \geq a) \wedge \neg P(z \geq a+1)$ と $\neg P(z \geq a) \vee P(z \geq a+1)$ で表せる．

4.2 alldifferent 制約の SAT 符号化

n 変数の alldifferent 制約 alldiff $\{z_1, \ldots, z_n\}$ は，以下のように $\frac{1}{2}n(n-1)$ 個の $\neq$ 制約で表せる．

$$\text{alldiff}\ \{z_1, \ldots, z_n\} \Longleftrightarrow \bigwedge_{1 \leq i < j \leq n} z_i \neq z_j$$

さらに，制約 $z \neq z'$ は以下のように表せる．

$$z \neq z' \Longleftrightarrow \bigwedge_{a \in \mathrm{Dom}(z) \cap \mathrm{Dom}(z')} (z \neq a \vee z' \neq a)$$

ここで $\mathrm{Dom}(z)$ と $\mathrm{Dom}(z')$ は整数変数 z と z' のドメインを表す．

整数変数 z と整数定数 a が等しくないことを表す制約 $z \neq a$ は，直接符号化の場合，以下の節に符号化できる．

$$z \neq a \Longleftrightarrow \neg P(z=a)$$

順序符号化の場合は，以下のように考えれば良い．

$$z \neq a \Longleftrightarrow \neg P(z \geq a) \vee P(z \geq a+1)$$

4.3 alldifferent 制約に対するヒント 1

順序符号化される alldifferent 制約に，**鳩の巣原理** を用いたヒントを加えると求解速度が向上することが知られている [14].

alldiff$\{z_1, \ldots, z_n\}$ について $z_i \in \{l, l+1, \ldots, u\}$ の時，以下の 2 節をヒントとして追加する．

$$\bigvee_{i=1}^{n} P(z_i \geq l+n-1)$$
$$\bigvee_{i=1}^{n} \neg P(z_i \geq u-n+1)$$

最初の節は，すべての z_i が $l+n-2$ 以下になることを禁じている．$l+n-2$ 以下の値は $n-1$ 通りしかないから n 個の互いに異なる値を割り当てることはできない．同様に，2 番目の節は，すべての z_i が $u-n+1$ 以上になることを禁じている．

4.4 alldifferent 制約に対するヒント 2

alldiff$\{z_1, \ldots, z_n\}$ について $z_i \in \{l, l+1, \ldots, u\}$ かつ $u - l = n - 1$ の時 (すなわち $|\mathrm{Dom}(z_i)| = n$ の時)，より有効なヒントを追加することができる．

このような場合，各値 $a \in \{l, l+1, \ldots, u\}$ に対し，その値を取る z_i が存在するから，以下が成り立つ．

$$\bigvee_{i=1}^{n} z_i = a \qquad (a \in \{l, l+1, \ldots, u\})$$

直接符号化の場合は，そのまま節に変換できる．

$$\bigvee_{i=1}^{n} P(z_i = a) \qquad (a \in \{l, l+1, \ldots, u\})$$

順序符号化の場合は **Tseitin 変換** と呼ばれる手法を用いる．すなわち $z_i = a$ を表す新しいブール変数 p_{ia} を導入し，p_{ia} を用いて条件を表す．

$$\bigvee_{i=1}^{n} p_{ia} \qquad (a \in \{l, l+1, \ldots, u\})$$

さらに $p_{ia} \Rightarrow z_i = a$ を意味する節を各 i と各 a に対し導入する．

$$\neg p_{ia} \vee P(z_i \geq a)$$
$$\neg p_{ia} \vee \neg P(z_i \geq a + 1)$$

5. 性能比較

ここまでで説明した制約モデルおよび SAT 符号化を比較する．比較対象は以下の 12 種類である．

(1) COL+D: 色変数モデル+直接符号化
(2) COL+D+H2: 色変数モデル+直接符号化+ヒント 2
(3) COL+O: 色変数モデル+順序符号化
(4) COL+O+H1: 色変数モデル+順序符号化+ヒント 1
(5) COL+O+H2: 色変数モデル+順序符号化+ヒント 2
(6) COL+O+H3: 色変数モデル+順序符号化+ヒント 1 と 2
(7) POS+D: 位置変数モデル+直接符号化
(8) POS+D+H2: 位置変数モデル+直接符号化+ヒント 2
(9) POS+O: 位置変数モデル+順序符号化
(10) POS+O+H1: 位置変数モデル+順序符号化+ヒント 1
(11) POS+O+H2: 位置変数モデル+順序符号化+ヒント 2
(12) POS+O+H3: 位置変数モデル+順序符号化+ヒント 1 と 2

これら 12 種類の方法を用い，7 から 13 の各 N に対し SAT 符号化して得られた命題論理式を SAT ソルバーで求解した時の CPU 時間を計測する．タイムアウトは 60 分とし，SAT ソルバーには GlueMiniSat version 2.2.8 [9] を用いた．

表 2 に，計測した CPU 時間を示す．TO は，タイムアウトの 60 分 (3600 秒) 以内に終わらなかったことを表す．残念ながら $N = 12, 13$ については，いずれの方法でも 60 分以内で解を求めることはできなかった．

alldifferent 制約に対するヒントがない制約モデル (COL+D, COL+O, POS+D, POS+O) を比較すると，位置変数モデル (POS) のほうが優れている．また，色変数モデル (COL) については順序符号化のほうが良い．

alldifferent 制約に対するヒントの有無で比較すると，ヒントは有効に働いていることがわかる．ヒントが加わった状態で見ると，位置変数モデル (POS) より色変数モデル (COL) のほうが良いようだ．

順序符号化に対する 3 種類のヒントのうち，どのヒントが最も有効かの判断は難しい．色変数モデルではヒント 1 と 2 の両方を加えた場合 (COL+O+H3)，位置変数モデルではヒント 2 を加えた場合 (POS+O+H2) が最も良いが，決定的ではない．

クイーングラフ彩色問題を少し変形した**汎対角ラテン方陣** (pandiagonal Latin square) についても実験を行った．汎対角ラテン方陣では，対角線の条件が汎対角線に変更される．すなわち $(i + j) \bmod N$ (あるいは $(i - j) \bmod N$) が等しいクイーン同士の色は異なる．したがって色変数モデルを用いた場合，すべての alldifferent 制約に

表 2 クイーングラフ彩色問題に対する CPU 時間 (秒)
(COL: 色変数モデル，POS: 位置変数モデル，D: 直接符号化，O: 順序符号化，H1: ヒント 1，H2: ヒント 2，H3: ヒント 1 と 2)

	$N=7$	$N=8$	$N=9$	$N=10$	$N=11$	$N=12$	$N=13$
	SAT	UNSAT	UNSAT	UNSAT	SAT	SAT	SAT
COL+D	0.0	14.4	TO	TO	TO	TO	TO
COL+D+H2	0.0	0.0	0.8	15.4	193.9	TO	TO
COL+O	0.0	11.0	240.6	1466.2	TO	TO	TO
COL+O+H1	0.0	0.0	620.4	TO	TO	TO	TO
COL+O+H2	0.0	0.0	2.4	113.2	3336.3	TO	TO
COL+O+H3	0.0	0.0	1.7	104.0	392.8	TO	TO
POS+D	0.0	0.0	2.0	37.3	3114.2	TO	TO
POS+D+H2	0.0	0.0	1.4	27.0	2893.5	TO	TO
POS+O	0.0	0.0	2.3	138.8	TO	TO	TO
POS+O+H1	0.0	0.0	2.3	114.4	969.7	TO	TO
POS+O+H2	0.0	0.0	2.7	89.2	783.9	TO	TO
POS+O+H3	0.0	0.0	2.4	100.4	TO	TO	TO

表 3 汎対角ラテン方陣問題に対する CPU 時間 (秒)
(COL: 色変数モデル，D: 直接符号化，O: 順序符号化，H1: ヒント 1，H2: ヒント 2，H3: ヒント 1 と 2)

	$N=7$	$N=8$	$N=9$	$N=10$	$N=11$	$N=12$	$N=13$
	SAT	UNSAT	UNSAT	UNSAT	SAT	UNSAT	SAT
COL+D	0.0	0.1	234.3	TO	TO	TO	TO
COL+D+H2	0.0	0.0	0.0	0.0	0.0	0.0	550.8
COL+O	0.0	0.1	24.4	TO	TO	TO	TO
COL+O+H1	0.0	0.0	0.0	0.0	620.3	0.1	TO
COL+O+H2	0.0	0.0	0.0	0.1	0.1	0.1	138.2
COL+O+H3	0.0	0.0	0.0	0.0	0.1	0.1	500.3

対してヒント 1 だけでなくヒント 2 も適用可能になる．

色変数モデルを用いた結果を表 3 に示す．クイーングラフ彩色問題とは異なり $N=12$ の時の解は存在しない．この問題に対しては，alldifferent 制約へのヒントが非常に有効に働いていることがわかる．

6. SAT 型制約ソルバー

ここまでで，問題の制約モデルを考え，それを SAT 符号化する方法について述べた．しかし，SAT 符号化を実際に行うのは面倒な作業である．

そのような作業を支援するシステムとして，**SAT 型制約ソルバー** (SAT-based constraint solver) があり，以下のようなものが提案されている．

- Sugar [*3] [17], Diet-Sugar [*4] [11], Copris [*5] [19], Scarab [*6] [12], BEE [8], meSAT [13]

特に著者らが開発している Sugar は，2008 年と 2009 年の制約ソルバー競技会 [7] の複数部門で優勝し，他の制約ソルバーに匹敵する性能を示した．SAT 符号化には順序符号化を利用している．Diet-Sugar では，順序符号化と対数符号化をハイブリッドさせ，さらなる性能向上を実現している．

以下は 5 次クイーングラフ彩色問題を Sugar および Diet-Sugar への入力として記述した例である．

[*3] http://bach.istc.kobe-u.ac.jp/sugar/
[*4] http://kix.istc.kobe-u.ac.jp/~soh/dsugar/
[*5] http://bach.istc.kobe-u.ac.jp/copris/
[*6] http://kix.istc.kobe-u.ac.jp/~soh/scarab/

```
import jp.kobe_u.copris._; import dsl._

object QG {
  def qg(n: Int) {
    val N = 0 until n
    for (i <- N; j <- N)
      int('c(i,j), 0, n-1)
    for (i <- N)
      add(Alldifferent(for (j <- N) yield 'c(i,j)))
    for (j <- N)
      add(Alldifferent(for (i <- N) yield 'c(i,j)))
    for (u <- 0 to 2*n-2)
      add(Alldifferent(for (i <- N; j <- N; if i+j==u) yield 'c(i,j)))
    for (d <- 1-n to n-1)
      add(Alldifferent(for (i <- N; j <- N; if i-j==d) yield 'c(i,j)))
    for (j <- N)
      add('c(0,j) === j)
    if (find) {
      for (i <- N) {
        val s = N.map(j => solution('c(i,j)))
        println(s.map("%2d".format(_)).mkString(" "))
      }
    }
  }
  def main(args: Array[String]) {
    qg(args(0).toInt)
  }
}
```

図 3 n 次クイーングラフ彩色問題の Copris でのプログラム例

```
(int c00 0 4)
(int c01 0 4)
..途中略..
(alldifferent c00 c01 c02 c03 c04)
(alldifferent c10 c11 c12 c13 c14)
..以下略..
```

(int c00 0 4) は，色変数 $c_{00} \in \{0,1,2,3,4\}$ の定義を表し，(alldifferent ...) は alldifferent 制約を表す．

上記入力を Sugar あるいは Diet-Sugar へ与えると，解として以下のような出力が得られる．

```
s SATISFIABLE
a c00   0
a c01   1
..以下略..
```

Copris および Scarab は，プログラミング言語 Scala 上に実現された制約プログラミング用のドメイン特化言語である．Scala が提供する高度な記述力により，わかりやすく簡潔に制約モデルを記述できる．図 3 に，クイーングラフ彩色問題の Copris でのプログラム例を示す．Copris は，バックエンドの制約ソルバーとして Sugar を用いており，Sugar と同等の性能を持つ．

7. おわりに

本稿ではクイーングラフ彩色問題を題材とし，制約モデル・SAT 符号化の選択肢について説明・比較を行った．

一般的なグラフ彩色問題に対しては，本稿で述べた色変数モデルと直接符号化の組合せが用いられることが多いと思われる．しかし，その性能は必ずしも良くなく，順序符号化のほうが良いことが多い [15][18]．Knuth の教科書 [6] の中でも，直接符号化よりも順序符号化のほうがグラフ彩色問

題に対して良い事が述べられている．

適切なヒントを加えれば，どちらの符号化でも性能が向上する．しかし，本稿の方法では N が 12 以上の場合の解を求めることに成功しなかった．読者諸氏による挑戦を期待したい．

参考文献

[1] 番原睦則，鍋島英知：SAT 技術の進化，情報処理， Vol. 57, No. 8, pp. 704–709 (2016).

[2] 番原睦則，宋 剛秀，田村直之，井上克巳：私のブックマーク：SAT ソルバー，人工知能学会誌，Vol. 28, No. 2, pp. 343–348 (2013).

[3] Biere, A., Heule, M., van Maaren, H. and Walsh, T.(eds.): *Handbook of Satisfiability*, IOS Press (2009).

[4] de Kleer, J.: A Comparison of ATMS and CSP Techniques, *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI 1989)*, pp. 290–296 (1989).

[5] 井上克巳，田村直之：SAT ソルバーの基礎，人工知能学会誌， Vol. 25, No. 1, pp. 57–67 (2010).

[6] Knuth, D. E.: *Satisfiability*, The Art of Computer Programming, Vol. 4, Fascicle 6, Addison-Wesley Professional (2015).

[7] Lecoutre, C., Roussel, O. and van Dongen, M. R. C.: Promoting Robust Black-box Solvers Through Competitions, *Constraints*, Vol. 15, No. 3, pp. 317–326 (2010).

[8] Metodi, A. and Codish, M.: Compiling finite domain constraints to SAT with BEE, *Theory and Practice of Logic Programming*, Vol. 12, No. 4-5, pp. 465–483 (2012).

[9] 鍋島英知，岩沼宏治，井上克巳：GlueMiniSat 2.2.5: 単位伝搬を促す学習節の積極的獲得戦略に基づく高速 SAT ソルバー，コンピュータソフトウェア，Vol. 29, No. 4, pp. 146–160 (2012).

[10] 鍋島英知，宋 剛秀：高速 SAT ソルバーの原理，人工知能学会誌， Vol. 25, No. 1, pp. 68–76 (2010).

[11] Soh, T., Banbara, M. and Tamura, N.: A Hybrid Encoding of CSP to SAT Integrating Order and Log Encodings, *Proceedings of the 27th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2015)*, pp. 421–428 (2015).

[12] Soh, T., Tamura, N. and Banbara, M.: Scarab: A Rapid Prototyping Tool for SAT-Based Constraint Programming Systems, *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing (SAT 2013), LNCS 7962*, pp. 429–436 (2013).

[13] Stojadinovic, M. and Maric, F.: meSAT: multiple encodings of CSP to SAT, *Constraints*, Vol. 19, No. 4, pp. 380–403 (2014).

[14] 田島宏史：SAT 変換に基づく制約ソルバーの高速化に関する研究，修士論文，神戸大学大学院工学研究科情報知能工学専攻 (2008).

[15] Tamura, N., Taga, A., Kitagawa, S. and Banbara, M.: Compiling Finite Linear CSP into SAT, *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP 2006), LNCS 4204*, pp. 590–603 (2006).

[16] Tamura, N., Taga, A., Kitagawa, S. and Banbara, M.: Compiling Finite Linear CSP into SAT, *Constraints*, Vol. 14, No. 2, pp. 254–272 (2009).

[17] 田村直之，丹生智也，番原睦則：SAT 変換に基づく制約ソルバーとその性能評価，コンピュータソフトウェア，Vol. 27, No. 4, pp. 183–196 (2010).

[18] 田村直之，丹生智也，番原睦則：制約最適化問題と SAT 符号化，人工知能学会誌，Vol. 25, No. 1, pp. 77–85 (2010).

[19] 田村直之，丹生智也，番原睦則：Scala 上の制約プログラミング用ドメイン特化言語 Copris について，コンピュータソフトウェア，Vol. 29, No. 4, pp. 114–129 (2012).