位取り記数法に基づく整数有限領域上の制約充足問題のコンパクトかつ効率的なSAT符号化

丹生，智也
田村，直之
番原，睦則

(Citation)
コンピュータソフトウェア,30(1):211-230

(Issue Date)
2013

(Resource Type)
journal article

(Version)
Version of Record

(URL)
https://hdl.handle.net/20.500.14094/90002834

位取り記数法に基づく整数有限領域上の制約充足問題のコンパクトかつ効率的なSAT符号化

丹生 智也　田村 直之　番原 睦則

本論文では，整数有限領域上の制約充足問題に現れる算術的な制約を SAT に符号化する新しい方法としてコンパクト順序符号化を提案する．コンパクト順序符号化の基本アイデアは，各整数変数を位取り基数法 (すなわち B 進法) で表現することと，各桁を順序符号化を用いて SAT に符号化することである ($B \geq 2$，B を基底と呼ぶ)．コンパクト順序符号化は $B = 2$ の場合には対数符号化と等価であり，B が整数変数のドメインサイズ以上の時には順序符号化と等価である．その意味で，コンパクト順序符号化は両方の符号化の一般化であるとみなせる．また，典型的な制約充足問題であるオープンショップ・スケジューリング問題とグラフ彩色問題とを用いた比較実験により，コンパクト順序符号化が小規模 (変数のドメインサイズが 10^2 未満) から大規模 (変数のドメインサイズが 10^7 程度) までの広い範囲の問題に適用可能であり，順序符号化および対数符号化，また既存の制約ソルバーよりも高性能であることが確認できた．今後，基底 B の選び方に関する網羅的な実験が必要と考えられるが，現時点でコンパクト順序符号化は，様々な規模の制約充足問題に適用可能かつ効率的な SAT 符号化の 1 つであるといえる．

This paper proposes a new SAT encoding method, named *compact order encoding*, which encodes arithmetic constraints used in Constraint Satisfaction Problems on finite integer domains. The basic idea of the compact order encoding is the use of a numeral system of some base $B \geq 2$ to represent integer variables, and each digit of variables is encoded by using the order encoding. It is equivalent the log encoding when $B = 2$, and it is equivalent to the order encoding when B is larger than the domain size of integer variables. Therefore, it can be seen as a generalization of the log and order encodings. We confirmed that the compact order encoding is applicable to wide range of problems from small-scale (where domain size of variables is less than 10^2) to large-scale (where domain size of variables is about 10^7), and shows better performance than log encoding, order encoding, and existing constraint solvers through experimental comparisons on Open- Shop Scheduling and Graph Coloring which are typical constraint satisfaction problems. From those results, although further comprehensive experiments will be necessary for selecting base B, it can be said that the compact order encoding is one of the effective SAT encodings at the current moment applicable to wide range of problems in various scales.

1　はじめに

命題論理式の充足可能性判定問題 (Boolean satisfiability testing; **SAT**) は，与えられた命題論理式を真にする値割り当てを探す組み合わせ問題である [6][24][51]．また **SAT 型システム**は，与えられた問題を SAT に符号化し，高速な SAT ソルバーを用いて元の問題の解を探索するシステムである (図 1)．近年，有界モデル検査 [5][4]，プランニング [28][38]，スケジューリング [10][44]，自動テストパターン生成 [15]，ソフトウェア検証 [29] 等の分野において，SAT 型システムは他の方法より優れた性能を示しており，問題解決の有望な方法として研究が行われている [6]．

特に制約充足の分野において，SAT 符号化に順序

A Compact and Efficient SAT Encoding of Finite CSP based on a Numeral System of Integers.
Tomoya Tanjo, 新領域融合研究センター, Transdisciplinary Research Integration Center.
Naoyuki Tamura, Mutsunori Banbara, 神戸大学情報基盤センター, Information Science and Technology Center, Kobe University.
コンピュータソフトウェア, Vol.30, No.1 (2013), pp.211–230.
[研究論文] 2011 年 10 月 20 日受付.
大会同時投稿論文

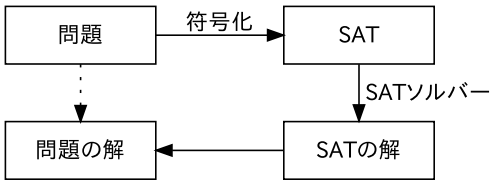

図 1　SAT 型システム

符号化を用いた SAT 型制約ソルバー Sugar[†1] [3] は，2008 年度と 2009 年度の国際制約ソルバー競技会[†2]の複数部門で優勝しており，既存の制約ソルバーと比較しても，SAT 型システムが有効であることを示している．また制約ソルバーと類似の擬ブール制約ソルバー，背景理論付き SAT (Satisfiability Modulo Theories; **SMT**) ソルバーにおいても，それぞれ Minisat+ [17]，MiniSmt [53] といった SAT 型システムが成功を収めている．

制約充足問題 (Constraint Satisfaction Problems; **CSP**) [39] から SAT への符号化は直接符号化 [13][52][45]，支持符号化 [27][19][45]，対数符号化 [25][18][45]，順序符号化 [43][44][45] 等数多く提案されているが，既存の符号化にはそれぞれ欠点がある．例えば，直接符号化と支持符号化は変数のドメインサイズを d とすると，各 3 項制約は $O(d^3)$ 個の節に符号化されるため，ドメインサイズが 10^2 以上あるような問題に対しては多くの節数が必要になり，実用上適用できない．また第 4.1 節で述べるように，順序符号化は SAT ソルバーの単位伝播によって制約ソルバーの範囲伝播を実現することができるため，少ない決定変数の選択回数で矛盾を検出することが期待できる．しかし，各 3 項制約が $O(d^2)$ 個の節に符号化されるため，直接符号化，支持符号化と同様にドメインサイズが 10^4 以上あるような問題に対しては実用上適用できない．対数符号化は整数の表現に 2 進法を用いるため，制約を非常に少ない節数に符号化することができる (加算制約の場合は $\log d$ に比例)．しかし単位伝播によって最上位ビットの範囲伝播しか実現できず，矛盾の検出に，桁数 $\log_2 d$ に比例する回数の決定変数を選択する必要があるため，一般に順序符号化と比較して効率が悪くなる可能性が高いという問題がある．

そこで本論文では，SAT 型制約ソルバーの実装で生じる上記の問題を解決するため，コンパクトかつ効率的な新しい SAT 符号化であるコンパクト順序符号化を提案する．コンパクト順序符号化は整数有限領域上の CSP に適用可能な SAT 符号化であり，各整数変数を位取り基数法 (すなわち B 進法) で表現する ($B \geq 2$，B を基底と呼ぶ)．また各桁を順序符号化を用いて符号化する．したがって，コンパクト順序符号化は基底 $B = 2$ の場合には対数符号化と等価であり，$B \geq d$ の場合には順序符号化と等価となる．その意味で，コンパクト順序符号化は両方の符号化の一般化であるとみなせる．

筆者らはコンパクト順序符号化を，各変数のドメインサイズが 10^2 未満の小規模な問題から，ドメインサイズが 10^7 程度の大規模な問題までの広い範囲の問題に適用可能であることを目指して設計した．ここで，CSP 中の制約の個数は 10^3 程度を想定し，SAT ソルバーは 10^7 以下の節を取り扱えると想定している．以下ではこの想定の上で，コンパクト順序符号化の設計方針について議論する．

本論文の構成は以下のようになる．まず第 2 節で SAT および SAT ソルバーについて述べた後，第 3 節で制約充足問題について述べる．第 4 節で既存の SAT 符号化である順序符号化，直接符号化，支持符号化，対数符号化の概要を述べ，その問題点を指摘した後で，第 5 節でコンパクト順序符号化の提案を行う．第 6 節では様々なドメインサイズのオープンショップ・スケジューリング問題やグラフ彩色問題を用いて，コンパクト順序符号化の有効性を検証する性能評価を行う．第 7 節で関連研究について述べた後，第 8 節で結論を述べる．

2　SAT と SAT ソルバー

2.1　SAT

SAT は通常，**連言標準形** (Conjunctive Normal Form; **CNF**) の論理式で与えられる．CNF 式は節の連言であり，節の集合として表される．**節**はリテラル

†1 http://bach.istc.kobe-u.ac.jp/sugar/
†2 http://www.cril.univ-artois.fr/CPAI09/

の選言であり，**リテラル**は命題変数 p またはその否定 $\neg p$ を表す．各命題変数は 1 または 0 の値をとり，それぞれ真と偽を表す．節がある割り当てのもとで**単位節**であるとは，節中の 1 つのリテラルの値が未割り当てで，他のリテラルの値が 0 である場合をいう．

以下では，SAT の各インスタンスを **SAT 問題**と呼ぶ．ある SAT 問題の論理式を充足する値割り当てが存在するとき，その SAT 問題は**充足可能**であるといい，そのときの値割り当てをその SAT 問題の**解**という．値割り当てが存在しないとき，その SAT 問題は**充足不能**であるという．

2.2 SAT ソルバーの発展と DPLL アルゴリズム

近年，SAT ソルバーの高速化技術の発展により，DPLL アルゴリズム [12] に基づいた系統的 SAT ソルバーの性能が著しく向上している [11] [41] [34] [33] [36]．

この節では，多くの近代的な SAT ソルバーが採用している DPLL アルゴリズムを以下の例を用いて説明する．

$$
\begin{aligned}
C_1 &: \quad p_1 \\
C_2 &: \quad \neg p_1 \vee \neg p_2 \\
C_3 &: \quad p_2 \vee p_3 \\
C_4 &: \quad \neg p_3 \vee p_4 \vee \neg p_5 \\
C_5 &: \quad \neg p_4 \vee p_6 \\
C_6 &: \quad \neg p_4 \vee \neg p_6
\end{aligned}
$$

DPLL アルゴリズムの初期状態では，すべての命題変数が未割り当ての状態である．まず，SAT 問題の節中に単位節がもしあれば，その単位節の未割り当てのリテラルに 1 を割り当てる．この例では C_1 が単位節なので，p_1 に 1 を割り当てる．この単位節による値の割り当てを**単位伝播**という．単位節がなくなるまで，DPLL アルゴリズムは単位伝播を繰り返す．この例では p_1 に 1 を割り当てたあと，単位伝播により $\neg p_2$ に 1，p_3 に 1 を割り当てる．単位節がなくなると，変数選択ヒューリスティックスに基づき未割り当ての変数を選択し，1 か 0 を割り当てて単位伝播を繰り返す．この時に選択した変数を**決定変数**という．この例では，決定変数に p_4 を選び，この変数に 1 を割り当てるとする．その結果，単位伝播により p_6 と $\neg p_6$ を含む単位節が同時に発生する．これを**矛盾**という．p_6 に 0 と 1 のどちらを割り当ててもこの SAT 問題は充足しないため，バックトラックが起こる．この例では，決定変数 p_4 を選んだ時点までバックトラックし，p_4 に 0 を割り当てて探索を続ける．これを，すべての変数に値を割り当てるか (充足可能な場合)，バックトラック先がなくなる (充足不能な場合) まで繰り返す．

単位伝播は DPLL アルゴリズムの基本的な処理であり，全処理時間の 7 割から 9 割を占めている [32]．また決定変数を数多く選択すると，探索中に行われるバックトラックの回数が多くなってしまうために，実行速度が低下すると考えられる．このため DPLL アルゴリズムを採用している SAT ソルバーを用いて高速な求解を行うためには，決定変数の選択回数を少なくすることが重要である．

また近代的な SAT ソルバーは DPLL に加え以下のような多くの高速化技術を用いており，10^6 個の変数と 10^7 個の節を含む巨大な SAT 問題であっても扱うことが可能である [42]．

- CDCL (Conflict Driven Clause Learning) [41] [31]
- 変数選択ヒューリスティックス [32]
- 非時間順バックトラック [31]
- 監視リテラル [32]
- リスタート戦略 [20]

以下では，DPLL アルゴリズムと上記の高速化技術を採用している近代的 SAT ソルバーを単に SAT ソルバーと呼ぶ．

3 制約充足問題 (CSP)

(整数有限領域上の) CSP は，与えられた制約を充足する変数への値割り当てを探索する組み合わせ問題であり，本論文では以下のように定義する．

定義 1 (制約充足問題). (整数有限領域上の) *CSP* は，以下を満たす組 (X, l, u, P, C) である．

- X は，整数変数の有限集合である．
- l は X から整数 $\mathbf{Z}$ への写像であり，整数変数の下限を表す．
- u は X から整数 $\mathbf{Z}$ への写像であり，整数変数の上限を表す．

- P は，命題変数の有限集合である．
- C は，X と P 上の制約の有限集合であり，制約の連言を表す．各制約は，整数上の加減乗除および論理比較，$\vee, \wedge, \neg$ 等の論理演算を用いて記述されている．

以下では整数有限領域上の CSP を単に CSP と呼ぶ．CSP の制約を満たす値割り当てが存在するとき，その CSP は**充足可能**であるといい，その時の値割り当てを**解**という．解が存在しないとき，その CSP は**充足不能**であるという．

CSP の各算術制約に関して，部分式に対して新しい変数を導入することで，CSP に含まれる算術制約をすべて 3 項制約 (2 項，単項を含む) に制限することができる．例えば制約 $x+y+z=w$ は，新しい変数 v を導入することで，$(v=x+y)\wedge(w=v+z)$ と表すことができる．また，除算制約は，乗算と加算を含む算術制約で置き換えることができる．例えば制約 $z=x/y$ は，新しい変数 w, r を導入することで，$(x=yw+r)\wedge(0\leq r)\wedge(r<y)$ と表すことができる．一般に n 項制約は，$O(n)$ 個の 3 項制約に還元できる．また部分式を新しい変数で置き換えても，制約伝播の結果には影響を与えないと考えられる．

また，整数変数 x の下限が 0 でない場合には，$x'=x-l(x)$ を満たす新しい整数変数 x' で x を置き換えることで，CSP に含まれる全ての整数変数の下限を 0 に固定することができる．この処理は，変数のドメインサイズを変更することなく行うことができる．以下では簡単化のため，すべての算術制約が除算を含まない 3 項制約で，整数変数の下限が 0 の CSP についてのみ考える．

また第 4 節で各 SAT 符号化の評価基準として用いるため，上記の変換により得られる 3 項制約のみからなる CSP の規模を，変数のドメインサイズを元に 4 種類に分類して議論する．なお第 1 節で述べたように，想定する制約数は 10^3 程度である．

- **小規模**: $\sim 10^2$ 程度
  2009 年度の国際制約ソルバー競技会で用いられた問題のうち，約 82% がこのドメインサイズの変数を含む問題であり，多くの制約ソルバーにとって標準的な規模であると考えられる．
- **中規模**: $10^2 \sim 10^4$ 程度
  同競技会において，約 16%がこの大きさのドメインサイズの変数を含んでおり，これらは制約ソルバーにとって比較的中規模な問題だと考えられる．
- **大規模**: $10^4 \sim 10^7$ 程度
  同競技会において，この大きさのドメインサイズの変数を含む問題は約 2% しかないため，既存の制約ソルバーにとっても大規模な問題であると考えられる．

4 既存の SAT 符号化の概要

以下では主な SAT 符号化である順序符号化，直接符号化，支持符号化，対数符号化の概要を述べ，それぞれの利点と欠点を説明する．例として，整数変数 $x, y \in \{0..4\}$ 上の制約集合 $\{x+1\leq y, x\geq 2, y\leq 2\}$ を用いる．これらの制約からなる CSP は充足不能である．また各符号化において，SAT ソルバーの単位伝播が矛盾を検出するまでの流れを示す．さらに，以下の点について各符号化の違いを考察する．

符号化後のサイズ:

各整数変数を符号化するのに必要な命題変数と節の数および，2 項制約，3 項制約を符号化するのに必要な節数を示す．また小規模，中規模，大規模な CSP (制約数は 10^3 とする) を SAT 符号化し，符号化後の節数と SAT ソルバーが扱える節数の上限として想定している 10^7 を比較する．

制約ソルバーの伝播アルゴリズムとの関係:

SAT ソルバーでの単位伝播と，制約ソルバーの伝播アルゴリズムとの関係について述べる．また，2 項制約間の矛盾を検出するのに必要な決定変数の選択回数について考察する．

4.1 順序符号化

順序符号化は，各整数変数 x とそのドメインの値 a に対して，$x\leq a$ を表す命題変数 $p(x\leq a)$ を導入する [43] [44] [45]．すなわち，各整数変数 x に対して命題変数 $p(x\leq 0)$，$p(x\leq 1)$，…，$p(x\leq u(x)-1)$ を導入する．$x\leq u(x)$ は恒真であるため，$p(x\leq u(x))$ は不要である．また，各命題変数の順序関係を表す以

表 1 充足する値割り当て

$p(x \le 0)$	$p(x \le 1)$	$p(x \le 2)$	$p(x \le 3)$	解釈
1	1	1	1	$x = 0$
0	1	1	1	$x = 1$
0	0	1	1	$x = 2$
0	0	0	1	$x = 3$
0	0	0	0	$x = 4$

下の節を追加する．

$$\neg p(x \le i) \vee p(x \le i+1) \quad (0 \le i < u(x) - 1)$$

例えば，整数変数 $x \in \{0..4\}$ は，命題変数 $p(x \le 0)$, $p(x \le 1)$, $p(x \le 2)$, $p(x \le 3)$ で表される．$x \le 4$ は恒真であるため，$p(x \le 4)$ は不要である．また，各命題変数の順序関係を表す以下の節を追加する．

$$\neg p(x \le 0) \vee p(x \le 1)$$
$$\neg p(x \le 1) \vee p(x \le 2)$$
$$\neg p(x \le 2) \vee p(x \le 3)$$

例えば，$\neg p(x \le 0) \vee p(x \le 1)$ は，x が 0 以下であれば x が 1 以下であることを表している．また，上の節を充足可能にする値割り当ては表 1 の 5 通りのみであり，それぞれ $x = 0$ から $x = 4$ に対応している．

制約については，制約に違反する範囲を符号化する．すなわち，$a_1 < x_1 \le b_1$，$a_2 < x_2 \le b_2$，…，$a_n < x_n \le b_n$ 内のすべての点 $(x_1, x_2, \ldots, x_n)$ が制約を違反している時，以下の節を追加する．

$$\begin{array}{ll} & p(x_1 \le a_1) \vee \neg p(x_1 \le b_1) \\ \vee & p(x_2 \le a_2) \vee \neg p(x_2 \le b_2) \\ \vee & \qquad \ldots \\ \vee & p(x_n \le a_n) \vee \neg p(x_n \le b_n) \end{array}$$

線形式を用いた線形制約については，より簡潔な符号化が可能である [44]．a_i を非零の整数定数，c を整数定数，x_i を整数変数とすると，制約 $\sum_{i=1}^{n} a_i x_i \le c$ は以下のように符号化される．

$$\bigwedge_{b_i} \bigvee_{i} (a_i x_i \le b_i)^{\#}$$

ここで b_i は $\sum_{i=1}^{n} b_i = c - n + 1$ と $-1 \le b_i \le u(a_i x_i)$ を満たすように動くとする．また変換 $()^{\#}$ は以下のように定義する．

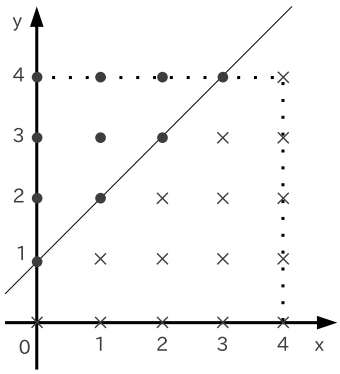

図 2 $x + 1 \le y$ の違反点

$$(ax \le b)^{\#} \equiv \begin{cases} p(x \le \lfloor b/a \rfloor) & (a > 0) \\ \neg p(x \le \lceil b/a \rceil - 1) & (a < 0) \end{cases}$$

ただし，$a < 0$ の場合には $p(x \le a)$ を 0 に変換し，最大値 $u(x)$ 以上については 1 に変換する．$x = y$ の形の制約は，$(x \le y) \wedge (y \le x)$ に置き換えることで，上の線形式に還元することができる．また非線形制約についても，上で述べた制約に違反する範囲を列挙する方法を用いて符号化できる．

例えば，制約 $x + 1 \le y$ は，以下の 5 つの節に符号化される (図 2 参照)．

$$\neg p(y \le 0)$$
$$p(x \le 0) \vee \neg p(y \le 1)$$
$$p(x \le 1) \vee \neg p(y \le 2)$$
$$p(x \le 2) \vee \neg p(y \le 3)$$
$$p(x \le 3)$$

例えば $p(x \le 0) \vee \neg p(y \le 1)$ は，$(x > 0) \wedge (y \le 1)$ が制約に違反する領域であることを表している．

また $x \ge 2$ と $y \le 2$ は，以下の節に符号化される．

$$\neg p(x \le 1) \qquad p(y \le 2)$$

これらの節から単位伝播により $\neg p(x \le 1)$，$\neg p(y \le 2)$，$p(y \le 2)$ が導出されるため，新たに決定変数を選択せずに制約集合 $\{x + 1 \le y, x \ge 2, y \le 2\}$ の矛盾が検出される．

符号化後のサイズ:

ドメインサイズを d とすると，各整数変数ごとに $O(d)$ 個の変数が導入される．また順序符号化では，加算，乗算等の制約の種類に関係なく，各 2 項制約は

$O(d)$ 個の節に，各 3 項制約は $O(d^2)$ 個の節に符号化される．したがって，小規模および中規模な問題に対しては適用可能であるが，それより大規模な問題に対しては 10^7 個を超える節が必要となるため，実質的に適用できないという問題がある．

制約ソルバーの伝播アルゴリズムとの関係:

SAT ソルバーでの単位伝播が制約ソルバーでの範囲伝播に対応しており，少ない決定変数の選択回数で矛盾を検出することが期待できる．さらに，直接符号化や対数符号化では実現できない tractable CSP (多項式時間で求解可能な CSP) から tractable SAT (多項式時間で求解可能な SAT) への符号化が順序符号化では可能であり，他の符号化よりも優れていることが理論的に示されている [37].

なお，整数変数の符号化に $p(x \leq a)$ を用いるものには unary 符号化 [2] や regular 符号化 [1] がある．unary 符号化は，一般の線形制約ではなく，$\sum x_i \geq a$ (x_i のドメインは $\{0,1\}$) 等の形に限定された制約のみを対象としている点で順序符号化とは異なる．regular 符号化では，整数変数の符号化において，後述する直接符号化を用いた場合に必要となる節数を減らす目的で $p(x \leq a)$ を用いている．しかし，制約の符号化自体は直接符号化で行っており順序符号化とは大きく異なる．$p(x \leq a)$ を制約の符号化に用いた研究としては，ジョブショップ・スケジューリング問題への適用例がある [10][23][35]．順序符号化は，これを CSP に適用可能なように一般化したものである．

4.2 直接符号化，支持符号化

直接符号化は最も広く用いられている SAT 符号化である．直接符号化は各整数変数 x とそのドメインの値 a に対して，$x = a$ を表す命題変数 $p(x = a)$ を用いる [13][52][45]．制約については，制約を違反点集合として表し，各点を禁止する節を追加する．支持符号化は変数の表現方法は直接符号化と同様だが，各制約の支持点集合に着目する点が異なる [27][19]．また支持符号化は 2 項制約に対してのみ適用できる．これらの方法は，加算や乗算等の制約の種類に関係なく適用できる．

まず，CSP の各整数変数は $O(d)$ 個の命題変数で表される．また，これらの符号化は，各 2 項制約を符号化するのに $O(d^2)$ 個の節が必要である．また直接符号化は，各 3 項制約を符号化するのに $O(d^3)$ 個の節が必要であるため，小規模な問題にしか適用できない．

直接符号化では，SAT ソルバーでの単位伝播により制約ソルバーでのフォワードチェック法が実現できる [52] が，これは範囲伝播よりも弱い制約伝播アルゴリズムである．支持符号化では単位伝播によりアーク性能維持法が実現できる [19]．これは範囲伝播よりも強いアルゴリズムである [8] が，支持符号化は順序符号化と比較して経験的に性能が良くないことが知られている [45].

4.3 対数符号化

対数符号化は，整数の 2 進法表記に着目した SAT 符号化である [25][18][45]．各整数変数 x の i 桁目 (LSB を 0 桁目とする) が 1 であることを表す命題変数 $p(x^{(i)} = 1)$ を導入する．

例えば，整数変数 $x \in \{0..4\}$ に対して，命題変数 $p(x^{(0)} = 1)$, $p(x^{(1)} = 1)$, $p(x^{(2)} = 1)$ を導入する．また，$x \leq 4$ を表す以下の節を追加する．

$$\neg p(x^{(2)} = 1) \vee \neg p(x^{(1)} = 1)$$
$$\neg p(x^{(2)} = 1) \vee \neg p(x^{(0)} = 1)$$

各節はそれぞれ,「$x^{(2)}$ が 1 ならば $x^{(1)}$ は 0」,「$x^{(2)}$ が 1 ならば $x^{(0)}$ は 0」を表している．

制約については，各桁ごとの制約を考え，各桁の制約に相当する論理回路を考えることで SAT に符号化する．例えば制約 $x + 1 \leq y$ に関して，各桁での制約は以下のように表される．ここで，c_1，c_2 はそれぞれ $x + 1$ の 1 桁目と 2 桁目の値からの桁上がりを表す新しい整数変数である．

$$x^{(2)} + c_2 \leq y^{(2)}$$
$$x^{(2)} + c_2 \leq y^{(2)} - 1 \vee x^{(1)} + c_1 \leq y^{(1)} + 2c_2$$
$$\cdots$$

各制約はそれぞれ,「$x+1$ の 2 桁目の値は $y^{(2)}$ 以下である」,「$x+1$ の 2 桁目の値が $y^{(2)}$ 以上ならば，$x+1$ の 1 桁目の値は $y^{(1)}$ 以下である」ことを表している．

この制約をそのまま SAT に符号化し，分配則等を用いて CNF 式に変換すると，節数が元の制約数に対

して指数関数的に増加してしまう．そこで，以下のように Tseitin 変換 [45] を行うことで，符号化後の節数を，元の制約数に比例した数に抑えることができる (q, r は新しい命題変数)．

$$x^{(2)} + c_2 \le y^{(2)}$$
$$q \vee r$$
$$\neg q \vee x^{(2)} + c_2 \le y^{(2)} - 1$$
$$\neg r \vee x^{(1)} + c_1 \le y^{(1)} + 2c_2$$
$$\dots$$

各制約に相当する論理回路を考えることで，$x+1 \le y$ を表す以下の節が得られる．

$$\neg p(y^{(2)}=1) \vee p(c_2=1) \quad p(x^{(2)}=1) \vee \neg p(y^{(2)}=1)$$
$$p(x^{(2)}=1) \vee p(c_2=1) \quad q \vee r$$
$$\neg q \vee \neg p(y^{(2)}=1) \quad \neg q \vee p(c_2=1)$$
$$\neg q \vee p(x^{(2)}=1) \quad \dots$$

また $x \ge 2$ と $y \le 2$ は，以下の節に符号化される．

$$p(x^{(2)} = 1) \vee p(x^{(1)} = 1)$$
$$\neg p(y^{(2)} = 1)$$
$$\neg p(y^{(1)} = 1) \vee \neg p(y^{(0)} = 1)$$

この例では単位伝播により，$\neg p(y^{(2)} = 1)$ が導出されるが，それ以上の伝播は起こらない．一般に，各制約間の矛盾の検出にはビット数に比例する数の決定変数を選択する必要がある．

乗算制約に関しても同様に，制約を表す論理回路を考えることで符号化できる．

符号化後のサイズ:

ドメインサイズを d とすると，各整数変数ごとに $O(\log d)$ 個の変数が導入される．各加算制約は $O(\log d)$ 個の節に符号化される．また乗算制約は $O(\log^2 d)$ 個の節に符号化される．したがって，大規模な問題に対しても 10^4 個の節しか必要としない．

制約ソルバーの伝播アルゴリズムとの関係:

SAT ソルバーでの単位伝播は，フォワード・チェック法よりも弱い制約伝播となり [52]，最上位ビット (Most Significant Bit; MSB) での範囲伝播に対応する．したがって，各制約間の矛盾を検出するためには，$\log_2 d$ に比例する回数の決定変数の選択が必要となり，順序符号化よりも効率が悪くなると考えられる．

4.4 既存の符号化の特徴およびその問題点

後述の表 2 に，各符号化のサイズと実現できる伝播アルゴリズムをまとめたものを示す．“2 項”，“3 項” は，整数変数のドメインサイズを d としたとき，各 2 項制約および各 3 項制約を符号化するのに必要な節数を表しており，“制約伝播” は，SAT ソルバーの単位伝播で制約ソルバーのどのような伝播アルゴリズムが実現できるかを示している．また，“対数 (加算)” および “対数 (乗算)” は対数符号化で加算制約，乗算制約を符号化した時の節数をそれぞれ表している．支持符号化は 2 項制約にのみ適用可能であるため，“3 項” の欄を “-” で表している．

5 コンパクト順序符号化

この節では，上記の問題を解決する新しい SAT 符号化であるコンパクト順序符号化を提案し，その概要について述べる．第 1 節で述べたように，コンパクト順序符号化の基本アイデアは以下の 2 点である [47] [48] [46] [49]．

- 各整数変数を位取り基数法を用いて表現する．すなわち，各整数変数 x を $\sum_{i=0}^{m-1} B^i x_i$ で表す ($B \ge 2$，$m = \lceil \log_B d \rceil$，すべての x_i に対し $0 \le x_i < B$)．以下では B を**基底**，m を**桁数**と呼ぶ．
- 各桁 x_i を順序符号化を用いて符号化する．

これにより，サイズと速度に関して以下の効果が期待できる．

符号化後のサイズ:

- ドメインサイズを d とすると，基底 B (すなわち，桁数 $m = \lceil \log_B d \rceil$) を選んだ場合，各整数変数ごとに $O(B \log_B d)$ 個の変数が導入される．また後述のように，各 2 項制約は $O(B \log_B d)$ 個の節に，各加算制約，乗算制約は $O(B^2 \log_B d)$ 個と $O(B^3 \log_B d + B^2 \log_B^2 d)$ 個の節にそれぞれ符号化される．このため，変数同士の乗算制約を含まない大規模な問題に対しては $m = 3$ を，乗算制約を含む場合でも $m = 6$ を選べば，符号化後の節数が約 $10^4 \sim 10^7$ 個となり，SAT ソルバーで十分取り扱えるサイズに収まる．
- また中規模な問題に関しては，乗算を含まない

場合には $m=2$ を，含む場合でも $m=3$ を選べば符号化後の節数が約 $10^4 \sim 10^7$ 個となり，順序符号化よりドメインサイズが大きい問題へ適用可能である．

- ドメインサイズが 10^{10} という超大規模な問題に対しても，乗算を含まない場合には $m=5$，含む場合であっても $m=10$ などを桁数として選ぶことで約 10^7 個の節で済むため，SAT ソルバーで十分取り扱うことができると考えられる．

制約ソルバーの伝播アルゴリズムとの関係:

- SAT ソルバーでの単位伝播により，最上位桁 (Most Significant Digit; MSD) での範囲伝播を実現できる．
- コンパクト順序符号化は各整数変数を $m = \lceil \log_B d \rceil$ 桁で表現する．$B > 2$ を選んだ場合，これは 対数符号化の $\lceil \log_2 d \rceil$ 桁よりも少なくなる．このため，矛盾を検出するために必要な決定変数の選択回数が少なく，より効率的である．

第 1 節でも述べたように，コンパクト順序符号化は基底 $B=2$ の場合には対数符号化と等価であり，$B \geq d$ の場合には順序符号化と等価となる．その意味で，コンパクト順序符号化は，両方の符号化の一般化となっている．

5.1 整数変数の符号化

上で述べたようにコンパクト順序符号化では，各整数変数 $x \in \{0..d-1\}$ を $\sum_{i=0}^{m-1} B^i x^{(i)}$ で表し，各 $x^{(i)}$ を順序符号化を用いて SAT に符号化する．

例えば基底 $B=3$ の場合，整数変数 $x \in \{0..4\}$ は，x の 0 桁目と 1 桁目を表す変数 $x^{(1)}, x^{(0)} \in \{0..2\}$ を考え，各桁をそれぞれ順序符号化することで以下の命題変数で表される．

$$p(x^{(1)} \leq 0) \quad p(x^{(1)} \leq 1)$$
$$p(x^{(0)} \leq 0) \quad p(x^{(0)} \leq 1)$$

また，各桁の順序符号化により以下の節が追加される．

$$\neg p(x^{(1)} \leq 0) \vee p(x^{(1)} \leq 1)$$
$$\neg p(x^{(0)} \leq 0) \vee p(x^{(0)} \leq 1)$$

5.2 制約の符号化

コンパクト順序符号化では，各制約を以下の 2 つの手順で符号化する．

- 桁ごとの制約に還元する．
- 桁ごとの制約を順序符号化する．

整数有限領域上の任意の算術制約は比較，加算，乗算の組み合わせで表現可能である．例えば，制約 $3x = y - z$ は，新しい変数 w を導入することで，$(w = 3x) \wedge (y = w + z)$ のように置き換えることができる．定数倍 $z = ay$ の形の制約は，変数同士の乗算と同様の方法で符号化できる．従って以下では比較制約 $x \leq y$，加算制約 $z = x + y$ と変数同士の乗算制約 $z = xy$ についてのみ説明し，基底に $B=10$ を選択した場合のコンパクト順序符号化の例を示す．

5.2.1 比較の符号化

比較は各桁ごとの比較を考えることで符号化できる．例えば，$x, y \in \{0..99\}$ 上の制約 $x \leq y$ は，まず整数 x, y を $x^{(1)}$，$x^{(0)}$, $y^{(1)}$，$y^{(0)}$ でそれぞれ置き換えた制約を考え，以下のような桁ごとの比較を表す制約に還元する．

$$x^{(1)} \leq y^{(1)}$$
$$x^{(1)} \leq y^{(1)} - 1 \vee x^{(0)} \leq y^{(0)}$$

各制約はそれぞれ，「$x^{(1)}$ は $y^{(1)}$ 以下である」，「$x^{(1)}$ が $y^{(1)}$ 以上ならば，$x^{(0)}$ は $y^{(0)}$ 以下である」ことを表している．

次に対数符号化の場合と同様に，Tseitin 変換を行い以下の制約に還元する (q, r は新しい命題変数)．

$$x^{(1)} \leq y^{(1)}$$
$$q \vee r$$
$$\neg q \vee x^{(1)} \leq y^{(1)} - 1$$
$$\neg r \vee x^{(0)} \leq y^{(0)}$$

最後に，各制約を第 4.1 節で述べた方法で順序符号化する．

$$p(x^{(1)} \leq 0) \vee \neg p(y^{(1)} \leq 0)$$
$$p(x^{(1)} \leq 1) \vee \neg p(y^{(1)} \leq 1)$$
$$\cdots$$
$$p(x^{(1)} \leq 9) \vee \neg p(y^{(1)} \leq 9)$$
$$q \vee r$$
$$\neg q \vee \neg p(y^{(1)} \leq 0)$$
$$\neg q \vee p(x^{(1)} \leq 0) \vee \neg p(y^{(1)} \leq 1)$$
$$\cdots$$
$$\neg q \vee p(x^{(1)} \leq 7) \vee \neg p(y^{(1)} \leq 8)$$

$$\neg q \vee p(x^{(1)} \leq 8)$$
$$\neg r \vee p(x^{(0)} \leq 0) \vee \neg p(y^{(0)} \leq 0)$$
$$\neg r \vee p(x^{(0)} \leq 1) \vee \neg p(y^{(0)} \leq 1)$$
$$\cdots$$
$$\neg r \vee p(x^{(0)} \leq 9) \vee \neg p(y^{(0)} \leq 9)$$

5.2.2 加算の符号化

加算は各桁ごとの加算を考えることで符号化できる．例えば，$x, z \in \{0..99\}$ 上の制約 $z = x + 26$ は，まず整数 x, z を $x^{(1)}$，$x^{(0)}$，$z^{(1)}$，$z^{(0)}$ でそれぞれ置き換えた制約を考え，以下のような桁ごとの加算を表す制約に還元する．

$$z^{(1)} = x^{(1)} + 2 + c \qquad 10c + z^{(0)} = x^{(0)} + 6$$
$$c \leq 0 \vee x^{(0)} + 6 \geq 10 \qquad c \geq 1 \vee x^{(0)} + 6 \leq 9$$

ここで c は，$x^{(0)} + 6$ の桁上がりを表す整数変数である．上 2 つの制約はそれぞれ，「$z^{(1)}$ は $x + 26$ の 1 桁目と等しい」，「$z^{(0)}$ は $x + 26$ の 0 桁目と等しい」ことを表している．下 2 つの制約は，「$x^{(0)} + 6 \geq 10$ かつその時に限り $c \geq 1$」を表している．これらの制約をそれぞれ第 4.1 節で述べた方法で順序符号化することで，$z = x + 26$ をコンパクト順序符号化できる．

5.2.3 乗算の符号化

乗算は各桁ごとの乗算を考えることで符号化できる．例えば，$z, x, y \in \{0..99\}$ 上の制約 $z = xy$ は，x の各桁に関する乗算を考えることで，以下の制約で表現できる．ここで，w_i は $x^{(i)}y$ を表す新しい整数変数である．

$$w_1 = x^{(1)}y \qquad w_0 = x^{(0)}y$$
$$z = 10w_1 + w_0$$

また各 $x^{(i)}$ は 0 から 9 までの値しか取らないため，各 $w_i = x^{(i)}y$ は以下のように表すことができる．

$$\bigwedge_{a=0}^{9} \Big((x^{(i)} \leq a - 1) \vee (x^{(i)} \geq a + 1) \vee (w_i = ay) \Big)$$

また，ay の計算は各 i に対して何度も行われるため，各 ay に対して新しい整数変数 y_a を導入する．これにより，$z = xy$ は，以下のように表現できる．

$$\bigwedge_{a=0}^{9} \Big((x^{(1)} \leq a - 1) \vee (x^{(1)} \geq a + 1) \vee (w_1 = y_a) \Big)$$
$$\wedge \bigwedge_{a=0}^{9} \Big((x^{(0)} \leq a - 1) \vee (x^{(0)} \geq a + 1) \vee (w_0 = y_a) \Big)$$

	$y^{(1)}$	$y^{(0)}$
$\times$		9
	v_{01}	v_{00}
v_{11}	v_{10}	
${y_9}^{(2)}$	${y_9}^{(1)}$	${y_9}^{(0)}$

図 3 $100{y_9}^{(2)} + 10{y_9}^{(1)} + {y_9}^{(0)} = 9(10y^{(1)} + y^{(0)})$ の筆算

$$\wedge\ z = 10w_1 + w_0 \wedge \bigwedge_{a=0}^{9} y_a = ay$$

各 $y_a = ay$ $(0 \leq a < B)$ に関しては筆算を考え，各計算を制約として表せばよい．例えば，$y_9 = 9y$ は以下の制約で表される (図 3 参照)．ここで，v_{ij} は部分積の各桁を表す新しい整数変数，c は $v_{01} + v_{10}$ からの桁上がりを表す整数変数である．

$$10v_{11} + v_{10} = 9y^{(1)} \qquad 10v_{01} + v_{00} = 9y^{(0)}$$
$${y_9}^{(2)} = v_{11} + c \qquad 10c + {y_9}^{(1)} = v_{01} + v_{10}$$
$${y_9}^{(0)} = v_{00}$$

それぞれの制約を順序符号化することで，制約 $z = xy$ をコンパクト順序符号化できる．$z = 10w_1 + w_0$ に関しては，各桁の左シフトと第 5.2.2 節で説明した方法を組み合わせることで符号化できる．

5.3 コンパクト順序符号化での単位伝播の例

第 4 節で用いた例 $x + 1 \leq y$，$x \geq 2$ と $y \leq 2$ を用いて，コンパクト順序符号化が矛盾を検出するまでの単位伝播の流れを説明する．基底 $B = 3$ を選んだ時，例 $x + 1 \leq y$ は，第 5.2.1 節の比較と加算の符号化を組み合わせることで，以下の節に符号化される．$p(c \leq 0)$ は，$x^{(0)} + 1$ からの桁上がりを表す整数変数 c が 0 以下であることを表す新しい命題変数である．

$$p(x^{(1)} \leq 0) \vee \neg p(y^{(1)} \leq 0)$$
$$p(x^{(1)} \leq 0) \vee p(c \leq 0)$$
$$\neg p(y^{(1)} \leq 0) \vee p(c \leq 0)$$
$$q \vee r$$
$$\neg q \vee p(x^{(1)} \leq 0)$$
$$\neg q \vee \neg p(y^{(1)} \leq 0)$$
$$\neg q \vee p(c \leq 0)$$
$$\neg r \vee p(x^{(0)} \leq 0) \vee \neg p(c \leq 0)$$
$$\neg r \vee p(x^{(0)} \leq 1) \vee \neg p(c \leq 0)$$

表 2　既存の符号化の特徴

符号化	2 項	3 項
直接	$O(d^2)$	$O(d^3)$
支持	$O(d^2)$	-
対数 (加算)	$O(\log d)$	$O(\log d)$
対数 (乗算)†3	$O(\log^2 d)$	$O(\log^2 d)$
順序	$O(d)$	$O(d^2)$

符号化	制約伝播
直接	フォワードチェック法
支持	アーク整合維持法
対数	MSB での範囲伝播
順序	範囲伝播

表 3　コンパクト順序符号化と既存の符号化との特徴の比較

符号化	2 項	3 項
直接	$O(d^2)$	$O(d^3)$
支持	$O(d^2)$	-
対数 (加算)	$O(\log d)$	$O(\log d)$
対数 (乗算)	$O(\log^2 d)$	$O(\log^2 d)$
順序	$O(d)$	$O(d^2)$
提案 (加算)	$O(B \log_B d)$	$O(B^2 \log_B d)$
提案 (乗算)	$O(B^2 \log_B^2 d)$	$O(B^3 \log_B d + B^2 \log_B^2 d)$

符号化	制約伝播
直接	フォワードチェック法
支持	アーク整合維持法
対数	MSB での範囲伝播
順序	範囲伝播
提案	MSD での範囲伝播

$$\neg r \vee \neg p(y^{(0)} \leq 0) \vee \neg p(c \leq 0)$$
$$\neg p(x^{(0)} \leq 1) \vee p(c \leq 0)$$
$$p(x^{(0)} \leq 1) \vee \neg p(c \leq 0)$$

また $x \geq 2$ と $y \leq 2$ は，以下の節に符号化される．

$$\neg p(x^{(1)} \leq 0) \vee \neg p(x^{(0)} \leq 1) \qquad p(y^{(1)} \leq 0)$$

この例の場合は，単位伝播により $p(y^{(1)} \leq 0)$, $p(c \leq 0)$, $p(x^{(1)} \leq 0)$, $\neg p(x^{(0)} \leq 1)$, $\neg p(c \leq 0)$ が導出されるため，決定変数を選択せずに矛盾を検出できる．また前述のように，一般的に各制約間の矛盾の検出には，$\log_B d$ に比例する回数の決定変数の選択しか必要としない．これは対数符号化の $\log_2 d$ 回より少なく効率的であると考えられる．

5.4　コンパクト順序符号化と他の符号化の比較

表 2 にコンパクト順序符号化を追加したものが表 3 である．“提案 (加算)” および “提案 (乗算)” はコンパクト順序符号化で加算制約，乗算制約を符号化した時の節数をそれぞれ表している．ここで B は基底を表す．

コンパクト順序符号化では，加算制約 $z = x + y$ は $O(B^2 \log_B d)$ 個の節に，乗算制約 $z = xy$ は $O(B^3 \log_B d + B^2 \log_B^2 d)$ 個の節に符号化される．特殊な場合として，$B \geq d$ の場合には $z = xy$ は $O(d^2)$ 個の節に符号化される．例えば $m = 2$ (すなわち $B = \sqrt{d}$) の場合，加算制約と乗算制約は $O(d)$, $O(d^{\frac{3}{2}})$ 個の節にそれぞれ符号化され，これは直接符号化，支持符号化，順序符号化の節数より少ない．

また SAT ソルバーの単位伝播により，MSD での範囲伝播を実現できる．例えば $d = 10^4$ の時，コンパクト順序符号化で $B = \sqrt{d}$ を基底に選んだ場合，各整数変数は 2 桁となるが，これは対数符号化の 14 桁よりはるかに少なく，矛盾の検出に必要な決定変数の選択回数が少なくなると期待できる．

6　性能評価

コンパクト順序符号化の有効性を検証するため，CSP の典型的な問題を用いた以下のベンチマークに対する性能評価を行った．

- **中規模かつ実際的な問題に対する性能評価:** 2006 年に SAT 型システムにより解かれるまで未解決であった問題を含むオープンショップ・スケジューリング問題 (OSS) を用いた．コンパクト順序符号化 ($m \in \{2, 3\}$) と既存の SAT 符号化の順序符号化，対数符号化，国際制約ソルバー競技会上位ソルバーである choco，Mistral の求解数および求解 CPU 時間を比較した．

†3　表では，筆算を符号化する方法での計算量を示している．FFT を用いることで，節数を $O(m \log m)$ まで減らすことができる．

表 4 OSS ベンチマークにおける求解 CPU 時間の比較 (単位: 秒)

問題	サイズ	順序	コンパクト順序		対数	choco	Mistral
			$m=2$	$m=3$			
j7-per0-0	7x7	T.O.	T.O.	T.O.	T.O.	T.O.	T.O.
j7-per0-1	7x7	56.16	**11.18**	16.06	119.52	T.O.	27.10
j7-per0-2	7x7	36.15	**8.35**	11.78	85.39	T.O.	49.92
j7-per10-0	7x7	56.01	**15.47**	17.88	100.07	T.O.	76.81
j7-per10-1	7x7	24.98	7.74	6.82	66.32	**0.53**	0.97
j7-per10-2	7x7	497.15	298.91	**253.07**	2804.06	T.O.	546.06
j7-per20-0	7x7	4.43	4.17	3.09	5.18	0.54	**0.12**
j7-per20-1	7x7	13.38	**5.54**	7.54	19.80	T.O.	16.82
j7-per20-2	7x7	24.38	**7.91**	9.61	32.37	T.O.	26.76
j8-per0-1	8x8	T.O.	T.O.	T.O.	T.O.	T.O.	T.O.
j8-per0-2	8x8	161.73	**42.61**	119.35	478.10	T.O.	142.14
j8-per10-0	8x8	345.09	**157.60**	276.06	T.O.	T.O.	308.73
j8-per10-1	8x8	10.20	T.O.	17.76	10.65	**0.69**	1.38
j8-per10-2	8x8	T.O.	**2305.73**	T.O.	T.O.	T.O.	T.O.
j8-per20-0	8x8	3.14	3.06	14.16	13.83	**0.69**	1.53
j8-per20-1	8x8	3.05	15.35	7.75	21.64	**0.70**	1.30
j8-per20-2	8x8	3.83	2.95	22.43	17.61	**0.68**	1.43
	#Solved		**14**	**14**	**14**	13	6	**14**

- **大規模かつ実際的な問題に対する性能評価:**
上の OSS から生成した大規模な OSS を用いて，コンパクト順序符号化 ($m \in \{2,3\}$) と上記と同じ各ソルバーについて求解数および求解 CPU 時間を比較した．またコンパクト順序符号化 ($m \in \{2,3\}$) と順序符号化，対数符号化について，矛盾が起きるまでに決定変数を選択した回数を比較した．さらに，コンパクト順序符号化 ($m=5$) と 対数符号化について，ドメインサイズが 10^{10} 程度の超大規模な OSS を用いて，求解 CPU 時間を比較した．
- **小規模かつコンパクト順序符号化が有効でないと考えられる問題:**
各制約が変数間の順序関係を考慮しない制約で表されており，コンパクト順序符号化が有効でないと考えられるグラフ彩色問題を用いた．コンパクト順序符号化 ($m \in \{2,3\}$) と既存の SAT 符号化の 順序符号化，対数符号化，直接符号化，支持符号化の求解数を比較した．

6.1 オープンショップ・スケジューリング問題

オープンショップ・スケジューリング問題 (Open-Shop Scheduling Problem; OSS) は n 個のジョブ J_1，J_2，...，J_n と n 個のマシン M_1，M_2，...，M_n から構成される．また，各ジョブ J_i は n 個のオペレーション O_{i1}，O_{i2}，...，O_{in} から構成される．オペレーション O_{ij} はマシン M_j で実行され，処理時間が p_{ij} かかる (p_{ij} は正の整数)．満たすべき制約としては，ジョブ J_i 上の任意の異なる 2 つのオペレーション O_{ij} と O_{ik} は，同時には処理できない，マシン M_j 上の任意の異なる 2 つのオペレーション O_{ij} と O_{kj} は，同時には処理できない，の 2 種類がある．決定問題としての OSS の目的は，与えられた総所要時間 (makespan) 内にすべてのジョブが完了できるか否か

表 5 OSS ベンチマークにおける各係数 c ごとの求解数の比較

係数 c	ドメインサイズ d	問題数	順序	コンパクト順序 $m=2$	コンパクト順序 $m=3$	対数	choco	Mistral
1	10^3	17	**14**	**14**	**14**	13	6	**14**
10	10^4	17	12	**13**	**13**	**13**	6	12
10^2	10^5	17	8	**13**	**13**	12	7	7
10^3	10^6	17	0	**14**	13	12	7	4
10^4	10^7	17	0	12	**13**	**13**	7	2
	Total	85	34	**66**	**66**	63	33	39

を決定することである．

ベンチマーク問題として，Brucker *et al.* [7] の中で最も大きな問題である “j7”(全 9 問)，“j8”(全 8 問) の全 17 問を用いた．これは，2006 年まで最適値が未知であった問題である “j7-per0-0”，“j8-per0-1”，“j8-per10-2” の 3 問を含んでいる．各問題はそれぞれ 7 ジョブ・7 マシン，8 ジョブ・8 マシンで構成される．各問題の総所要時間として，最も難しい場合である最適値から 1 を引いたものを用いた (充足不能)．また各問題は，$x + a \leq z$ と $x \leq y$ の形の制約を用いて CSP で表現される．

コンパクト順序符号化 ($m \in \{2, 3\}$) と順序符号化，対数符号化について，SAT ソルバーが問題を解く (充足不能を証明する) までにかかった CPU 時間を比較した．また最先端の制約ソルバーである choco 2.11 [50] と Mistral 1.550 [21] とも比較を行った．

表 4 に制限時間 1 時間 (3600 秒)，MiniSat 2.0 core [16] ソルバーが求解にかかった時間を示す (単位は秒)．実行環境として Intel Xeon 3.0 GHz, メモリ 16GB のマシンを用いた．表中の “T.O.” は問題が 1 時間以内に解けなかったことを表す．表下の “#solved” は，解けた問題数を表す．各問題で最も速く解けたものを太字で表す．

コンパクト順序符号化 ($m \in \{2, 3\}$) は，順序符号化，Mistral と同様に，17 問中最も多い 14 問を解いた．またコンパクト順序符号化 ($m = 2$) は全 17 問中 8 問に関して，最も速く問題を解いている．更に，コンパクト順序符号化 ($m = 2$) のみが，2006 年まで未解決だった難しい問題である “j8-per10-2” の求解に成功している．

以上から中規模かつ実際的な問題の一例としてオープンショップ・スケジューリング問題について比較実験を行った結果，コンパクト順序符号化は順序符号化，対数符号化および既存の制約ソルバーと比較して高速に求解が行えることを示した．

6.2 大規模な OSS

大規模かつ実際的な問題でのコンパクト順序符号化の有効性を検証するため，先ほど用いた各 OSS 問題の処理時間を c 倍して生成した 102 問を用いて性能評価を行った (c は定数)．係数 c には 10^n ($n \in \{0..4\}$) を用いた．先ほどの実験と同様に，4 種類の符号化と choco，Mistral を比較した．

表 5 に各ソルバーの求解問題数を示す．表中の “ドメインサイズ d” は，各問題の大まかなドメインサイズを表し，各 c に対して最も解けた問題数が多いものを太字で示している．また図 4 に，各ソルバーでの求解 CPU 時間の比較を示す．横軸は求解問題数，縦軸に求解 CPU 時間を表している．コンパクト順序符号化 ($m = 2$) を細い実線で示し，コンパクト順序符号化 ($m = 3$) の場合を太い点線で示している．表 6 に，各 c での各符号化での平均求解時間を示す．$c \leq 10^2$ の場合には全ての符号化で解けた問題の平均求解時間，$c \geq 10^3$ の場合には順序符号化以外の全ての符号化で解けた問題の平均求解時間を示している．第 3 節で述べたように，ドメインサイズが $10^3 \leq d < 10^4$ の問題は中規模であり，$10^4 \leq d \leq 10^7$ の問題は大規模である．

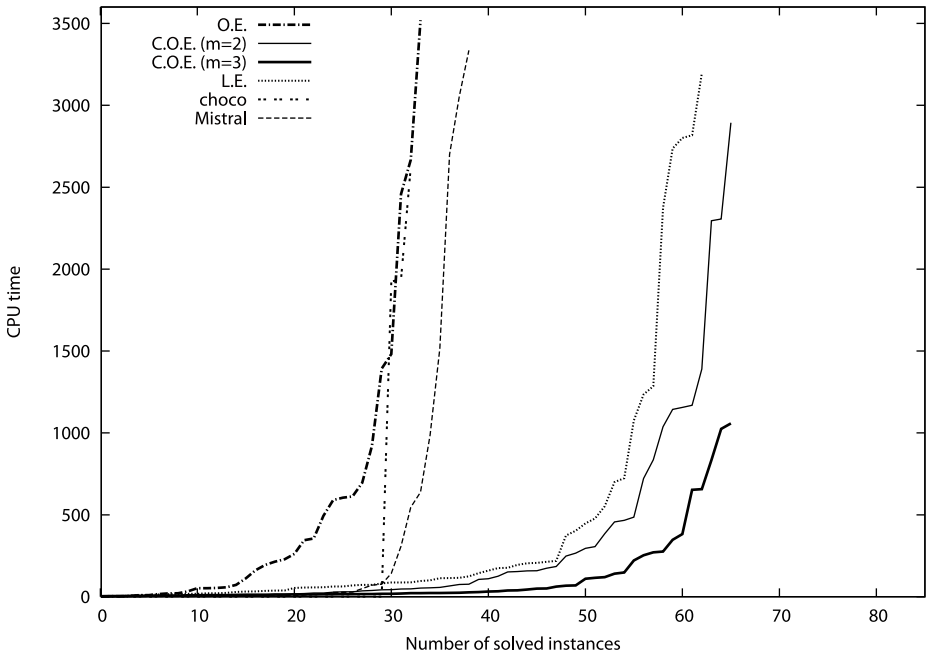

図 4 OSS ベンチマークにおける求解 CPU 時間の比較

表 6 OSS ベンチマークにおけるドメインサイズごとの平均求解時間 (単位: 秒)

係数 c	ドメインサイズ d	問題数	順序	コンパクト順序 $m=2$	コンパクト順序 $m=3$	対数
1	10^3	12	73.70	35.27	40.80	313.66
10	10^4	11	347.98	21.47	16.44	189.13
10^2	10^5	7	12276.30	43.49	17.08	53.08
10^3	10^6	11	-	93.48	33.46	203.41
10^4	10^7	11	-	704.43	77.66	398.60

コンパクト順序符号化 ($m \in \{2,3\}$) が 85 問中最も多い 66 問を解いた．またほとんどの制限時間において，コンパクト順序符号化 ($m=3$) が最も求解数が多くなっている．例えば制限時間が 600 秒の場合，コンパクト順序符号化 ($m=3$) は 61 問であり，対数符号化の 53 問を含め他ソルバーより多くの問題を解いている．

以下では，コンパクト順序符号化 ($m=3$) と各 SAT 符号化および既存の制約ソルバーを比較する．

コンパクト順序符号化 ($m=3$) と順序符号化を比較すると，コンパクト順序符号化 ($m=3$) は順序符号化よりも 30 問近く多くの問題を解いた．また順序符号化は $d \approx 10^3$ の場合には求解数および平均求解 CPU 時間はコンパクト順序符号化 ($m=3$) の数倍程度であるが，それより大きな d では求解数および求解 CPU 時間が悪化し，$d \geq 10^6$ の場合にはメモリ不足が原因で問題を 1 問も解くことができなかった．それに対してコンパクト順序符号化 ($m=3$) は，$d \approx 10^7$ の場合でも $d \approx 10^3$ の場合と比べてほとんど性能が落ちていない．このことから，コンパクト順序符号化 ($m=3$) は順序符号化には適用できない大規模な問題に対しても適用可能である．

コンパクト順序符号化 ($m=3$) と対数符号化を比較すると，すべての c についてコンパクト順序符号化

表 7 OSS ベンチマークにおける矛盾が起きるまでに選択した決定変数の数

係数 c	ドメインサイズ d	順序	コンパクト順序 $m=2$	コンパクト順序 $m=3$	対数
1	10^3	1.19	1.66	3.60	8.92
10	10^4	1.22	1.46	3.37	10.08
10^2	10^5	1.29	1.57	2.33	11.34
10^3	10^6	-	1.27	1.61	11.37
10^4	10^7	-	1.68	1.76	9.95

($m=3$) は対数符号化より求解数が多い．求解 CPU 時間の面に関しても，すべての c についてコンパクト順序符号化 ($m=3$) のほうが速い．例えば $c=10^4$ ($d\approx 10^7$) の時，コンパクト順序符号化 ($m=3$) は対数符号化の約 5 倍高速である．

次にコンパクト順序符号化 ($m=3$) と choco を比較すると，コンパクト順序符号化 ($m=3$) はすべての c について 6 問以上求解数が多い．また図 4 から，求解 CPU 時間に関してもコンパクト順序符号化 ($m=3$) のほうが高速である．このことから，中規模から大規模の問題ではコンパクト順序符号化 ($m=3$) は choco よりも優れているといえる．

コンパクト順序符号化 ($m=3$) と Mistral を比較すると，コンパクト順序符号化 ($m=3$) はすべての c について Mistral より多くの問題を解いている．また図 4 から，求解 CPU 時間に関してもコンパクト順序符号化 ($m=3$) は Mistral よりも高速である．このことから，中規模から大規模の問題ではコンパクト順序符号化 ($m=3$) は Mistral よりも優れているといえる．

次に各符号化の矛盾の検出能力を調べるため，表 7 に，各符号化での 1 つの矛盾を検出するまでに選択された決定変数の個数の平均値 (decision 数/conflict 数) を示す．$c\leq 10^2$ の場合にはいずれかの符号化で解けた問題に関する平均を示し，$c\geq 10^3$ の場合には順序符号化を除くいずれかの符号化で解けた問題に関して平均を示した．“-” は，1 問も問題を解けなかったものを表す．

いずれの c においても，桁数 m が少ないほど，矛盾が起きるまでに決定変数を選択する回数が少ない．特に $c=10^3$ の場合，コンパクト順序符号化 ($m=2$) での決定変数を選択する回数は，対数符号化の約 10 分の 1 であり，コンパクト順序符号化 ($m=2$) が対数符号化よりも矛盾の検出を早期に行えることがわかる．

以下にコンパクト順序符号化に関して，符号化にかかった時間と符号化後のサイズについて簡単に述べる．OSS に関して，コンパクト順序符号化の各インスタンスは，$c=10^4$ のときでも全て 15 秒以内に符号化できた．生成される SAT 問題のファイルサイズは，順序符号化は $c=10^3$ の時点で 16GB 以上の SAT 問題を生成したのに対し，コンパクト順序符号化 ($m=3$) は $c=10^4$ の場合でも 650MB 程度だった．

また，$c=10^i$ 以外のいくつかの係数についても性能評価を行い，求解数や求解 CPU 時間等に関して同様の結果となることを確認した．

最後に，OSS ベンチマークの “j7-per0-1” の各処理時間を 10^7 倍して超大規模の OSS を作成した．この OSS は，順序符号化は符号化後の SAT 問題のサイズが大きすぎて解くことができず，choco および Mistral は 10^{10} のドメインサイズの整数変数を扱えないため，この問題を解くことができない．第 4 節で用いたのと同じ目安だと，$m=5$ を選択した場合は節数が 10^7 個以下になる．コンパクト順序符号化 ($m=5$) と対数符号化を用いて求解し，求解 CPU 時間を比較した．結果，コンパクト順序符号化 ($m=5$) は約 1 分以内に求解できたが，対数符号化は求解に約 10 分とコンパクト順序符号化 ($m=5$) の 10 倍の時間が掛かった．

以上のように，大規模かつ実際的な問題の一例とし

表 8 GCP ベンチマークにおける求解数の比較

	問題数	直接	支持	順序	コンパクト順序		対数
					$m=2$	$m=3$	
求解数	104	88	85	91	89	90	90

て，オープンショップ・スケジューリング問題について比較実験を行った結果，コンパクト順序符号化が対数符号化，順序符号化よりも優れた結果となることがわかった．特に，ドメインサイズが 10^{10} 程度の超大規模な問題に対してもコンパクト順序符号化が適用可能であり，対数符号化よりも高速な求解が行えることを示した．

6.3 グラフ彩色問題

最後に小規模グラフ彩色問題を用いて，コンパクト順序符号化 ($m \in \{2,3\}$) と既存の SAT 符号化の 順序符号化，対数符号化，直接符号化，支持符号化を求解数に関して比較した．

グラフ彩色問題 (Graph Coloring Problem; GCP) は，頂点の有限集合 V と辺の有限集合 E からなるグラフ $G(V,E)$ で構成される．決定問題としての GCP の目的は，色数 k が与えられたときに各辺の両端が異なるように頂点を k 色で彩色できるかを判定する問題である．グラフが k 色以下で彩色可能なとき，そのグラフは **k-彩色可能**であるといい，この時の k を**彩色数**という．この問題は，ドメインサイズ k の整数変数が $|V|$ 個，$x \neq y$ の形の制約 $|E|$ 個からなる CSP で表現される．GCP は変数間の順序関係を考慮しない制約からなる問題であり，コンパクト順序符号化に不向きな問題だと考えられる．

ベンチマーク問題として，Graph Coloring and its Generalization [†4]で公開されている GCP 全 119 問のうち，彩色数 (最小色数) が既知の 52 問を用いた [†5]．これらの問題の頂点数は 11〜1085，辺数は 20〜121275，彩色数は 4〜63 である．52 問のベンチマーク問題に対して，コンパクト順序符号化 ($m \in \{2,3\}$)，直接符号化，支持符号化，順序符号化，対数符号化，直接符号化を用いて，k=彩色数 の場合 (充足可能) と k=彩色数 -1 の場合 (充足不能) の 2 種類の SAT 問題を生成した．

生成した SAT 問題全 728 問を MiniSat 2.0 core を用いて求解し，求解数を比較した．制限時間は 30 分 (1800 秒) で，実行環境は OSS の時と同様である．

表 8 に求解数を示す．順序関係のない問題であっても，順序符号化は 104 問中最も多い 91 問を解いた．コンパクト順序符号化 ($m \in \{2,3\}$) も，順序符号化との求解数の差は 2 問程度に収まっており，直接符号化や支持符号化よりも多くの問題を求解している．

また順序符号化，コンパクト順序符号化 ($m \in \{2,3\}$) と対数符号化で解けた問題のうち，コンパクト順序符号化 ($m=3$) と対数符号化の符号化の結果が異なる 20 問について平均 CPU 時間を比較した．コンパクト順序符号化 ($m=2$) の平均 CPU 時間は約 60.07 秒で，順序符号化の約 9.44 秒よりも約 6 倍遅く，コンパクト順序符号化 ($m=3$) の約 132.14 秒，対数符号化の約 149.23 秒よりも 2 倍以上高速だった．これは第 5.4 節で述べたように，桁数が少ないほうが，矛盾を検出するのに必要な決定変数の選択回数が少なくなるためだと考えられる．また順序符号化が直接符号化より求解数が多いのは，SAT ソルバーの学習節による効果だと考えられる．符号化後の問題サイズは，順序符号化が最大で約 300MB であったのに対して，コンパクト順序符号化は $m=2$ の場合でも，高々 83MB 程度であった．

以上から，小規模かつコンパクト順序符号化が有効でないと考えられるグラフ彩色問題についても，コンパクト順序符号化の性能がそれほど悪化しないことが確認できた．

6.4 考察

上記の実験結果から，一部の制約充足問題についてではあるが，コンパクト順序符号化が小規模から大

†4 http://mat.gsia.cmu.edu/COLOR04/

†5 各符号化による予備実験で最適値が決定できたものを含む．

規模の広い範囲の問題に適用可能であることがわかった．特に，ドメインサイズが 10^{10} 程度の超大規模な問題 1 問について，コンパクト順序符号化は対数符号化よりも 10 倍程度高速だった．また，小規模かつコンパクト順序符号化が有効でないと考えられるグラフ彩色問題についても，コンパクト順序符号化の性能がそれほど悪化しないことが確認できた．

コンパクト順序符号化の桁数 m (あるいは基底 B) の選び方については，中規模な問題に関しては桁数 $m = 2$，大規模な問題に関しては桁数 $m = 3$ とする基底が最も良い結果であった．したがって，$1 \leq m \leq 3$ の範囲からの選択で，小規模から大規模までの広い範囲の問題に適用可能だと考えている．ただし，このことはより広範な実験により確認する必要がある．

7 関連研究

Sugar 以外の SAT 型制約ソルバーとしては，FznTini [22] と SAT4J CSP [30] がある．FznTini は対数符号化を，SAT4J CSP は直接符号化と支持符号化の変種を用いている．

また，制約ソルバーと類似のものとして擬ブール制約ソルバーと SMT ソルバーがある．擬ブール制約は，各変数のドメインが $\{0, 1\}$ に制限された CSP の一種であり [40]，各整数変数を対数符号化と同様に 2 進法で表現することで，整数上の制約も表現可能である．Eén と Sörensson は，擬ブール制約の SAT 符号化に基数を可変にした位取り基数法を用い，各桁の表現に順序符号化と同様の 1 進法を用いる方法を提案している [17]．また，最適な基底を選択するアルゴリズムが Codish らにより提案されている [9]．これらの方法は，個々の制約において コンパクト順序符号化と同様の最上位桁での範囲伝播が期待できる．しかし，各変数は 2 進法で表現されるため，複数の制約にまたがった変数についての範囲伝播が実現できない．例えば，制約 $x + 50 \leq y$，$y + 100 \leq z$，$z \leq 149$ $(x, y, z \in \{0..200\})$ において，コンパクト順序符号化は第 5.3 節で示したように単位伝播のみで矛盾を検出できるが，Eén と Codish らの符号化方法の場合，矛盾を検出するには 2 進法の桁数に比例する個数の決定変数を選択する必要が生じ，この点がコンパクト順序符号化とは異なる．

SMT ソルバーは，SAT ソルバーを算術や配列等を取り扱う背景理論上のソルバーを組み合わせ，整数上の線形および非線形制約の問題を高速に解くことを実現している [26] [51]．特に MiniSmt は，制約式を単純に対数符号化する SAT 型のソルバーであるが，SMT ソルバー競技会の整数上の非線形制約部門 (QF_NIA) の問題に対し良い成績を示し [53]，2010 年度の競技会では QF_NIA を含む複数部門で優勝している．そこで，QF_NIA 部門の calypto 問題全 138 問のうち，ドメインサイズが 2^{63} (約 10^{19}) 以下である 129 問を用いて比較実験を行った．コンパクト順序符号化を用いてドメインサイズが 10^7 個を大きく超える問題を解く場合，$m = 3$ よりも大きな m を選ぶことで，符号化後の節数を 10^7 個以下に抑えることができる．例えば calypto 問題に対しては，$m = 15$ を選択した場合には節数が 10^7 個以下になる．比較実験の結果，求解数に関してはコンパクト順序符号化 ($m = 15$) は 103 問，MiniSmt は 116 問と 13 問少なく，求解 CPU 時間に関してはコンパクト順序符号化 ($m = 15$) は約 6.14 秒，MiniSmt は約 1 秒と約 6 倍程度遅いという結果となった．また，他の SMT ソルバーとの性能を比べるため，2011 年度 SMT ソルバー競技会中のオープンショップ・スケジューリング問題 (QF_IDL 部門) での結果との比較を行った．表 4 中と同一の問題である “j8-per-0-2” において，コンパクト順序符号化 ($m = 2$) は 2 位相当の結果となった (CPU クロック数の補正後)．また，MiniSmt と第 6.1 節の OSS を用いた比較実験では，MiniSmt は 3600 秒以内に 1 問も解くことができなかった．

次に，QF_IDL および QF_NIA 部門を含めた複数部門で優勝した Z3 [14] とコンパクト順序符号化を同一の実行環境で比較実験を行った．第 6.1 節で用いられた OSS による性能評価では，また，Z3 は与えられた 17 問全ての問題の求解に成功し，コンパクト順序符号化よりも良い性能を示した．しかし，15 ジョブ・15 マシンのジョブショップ・スケジューリング問題 5 問を用いた性能比較実験の結果では，コンパクト順序符号化 ($m = 2$) は平均求解 CPU 時間は 40.20 秒と，Z3 の 1032.53 秒よりも 25 倍以上高速であっ

た．これらの結果から，非線形制約を含む問題での性能については課題が残されているが，オープンショップ・スケジューリング問題およびジョブショップ・スケジューリング問題を用いた比較実験の結果については，コンパクト順序符号化は最新の SMT ソルバーと比較しても遜色ない性能であると言える．

8 結論

本論文では，整数有限領域上の制約充足問題のコンパクトかつ効率的な新しい SAT 符号化であるコンパクト順序符号化を提案した．コンパクト順序符号化の基本アイデアは，各整数変数を位取り基数法 (すなわち B 進法) で表現することと，各桁を順序符号化を用いて SAT に符号化することである ($B \geq 2$，B を基底と呼ぶ)．コンパクト順序符号化は $B = 2$ の場合には対数符号化と等価であり，B がドメインサイズ以上の時には順序符号化と等価である．その意味で，コンパクト順序符号化は両方の符号化の一般化であるとみなせる．

コンパクト順序符号化は小規模 (変数のドメインサイズが 10^2 未満) から大規模 (ドメインサイズが 10^7 程度) までの広い範囲の問題に適用可能である．このことを，限定的なベンチマークではあるが，様々なドメインサイズのオープンショップ・スケジューリング問題による実験結果により確認した．またオープンショップ・スケジューリング問題 1 問を用いた実験結果により，ドメインサイズが 10^{10} 程度の超大規模な問題に対しても，コンパクト順序符号化が適用できることを示した．

コンパクト順序符号化の基底 B (あるいは桁数 m) に関しては，CSP に含まれる各変数のドメインサイズや制約数等から自動的に選択が可能であり，そのように B (あるいは m) を選んだ場合でも，順序符号化および対数符号化，また既存の制約ソルバーと比較しても十分優れた性能が実現可能であることがわかった．

既存の SAT 符号化である順序符号化，対数符号化，最先端の制約ソルバーである choco，Mistral と比較した結果，コンパクト順序符号化は中規模な問題では $m = 2$，大規模な問題では $m = 3$ で優れた性能を示した．例えば中規模および大規模な問題 85 問中，コンパクト順序符号化 ($m \in \{2, 3\}$) は 66 問を解いたのに対し，順序符号化，choco，Mistral は 40 問以下であった．対数符号化の求解数は 63 問だが，平均求解時間を比較するとコンパクト順序符号化 ($m = 3$) のほうが約 5 倍高速であった．また，制限時間を 600 秒にすると，コンパクト順序符号化 ($m = 3$) は 61 問，対数符号化は 53 問とコンパクト順序符号化のほうがより多くの問題を解いた．

また，詳細についてはより広範な実験により確認する必要があるが，コンパクト順序符号化の桁数 m (あるいは基底 B) の選び方については，$1 \leq m \leq 3$ が小規模から大規模までの広い範囲の問題に有効だと考えている．

最後に，コンパクト順序符号化は大規模なドメインサイズの制約充足問題に対して適用可能かつ効率的な SAT 符号化であり，これにより SAT 型システムの適用範囲を更に広げることができると考えている．今後の課題としては，非線形制約を含む問題における性能改善や，擬ブール制約ソルバー，SMT ソルバーとの性能比較，桁数 m (あるいは基底 B) の選択方法についての検証を行うことが挙げられる．

謝辞

本論文の執筆にあたり，有用な御助言を賜りました査読者の方々に感謝致します．

参 考 文 献

[1] Ansótegui, C. and Manyà, F.: Mapping Problems with Finite-Domain Variables into Problems with Boolean Variables, in *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing* (*SAT 2004*), LNCS 3542, Springer-Verlag, 2004, pp. 1–15.

[2] Bailleux, O. and Boufkhad, Y.: Efficient CNF Encoding of Boolean Cardinality Constraints, in *Proceedings of the 9th International Joint Conference on Principles and Practice of Constraint Programming* (*CP 2003*), LNCS 2833, Springer-Verlag, 2003, pp. 108–122.

[3] 番原睦則, 丹生智也, 田村直之: SAT 変換に基づく制約ソルバー Sugar, 第 11 回プログラミングおよびプログラミング言語ワークショップ (PPL2009), 2009.

[4] Biere, A.: Bounded Model Checking, *Handbook of Satisfiability*, IOS Press, 2009, pp. 457–481.

[5] Biere, A., Cimatti, A., Clarke, E. M. and Zhu, Y.: Symbolic Model Checking without BDDs, in *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems* (*TACAS 1999*), LNCS 1579, Springer-Verlag, 1999, pp. 193–207.

[6] Biere, A., Heule, M., van Maaren, H. and Walsh, T.(eds.): *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications (FAIA), Vol. 185, IOS Press, 2009.

[7] Brucker, P., Hurink, J., Jurisch, B. and Wöstmann, B.: A Branch & Bound Algorithm for the Open-shop Problem, *Discrete Applied Mathematics*, Vol. 76, No. 1–3(1997), pp. 43–59.

[8] Choi, C. W., Harvey, W., Lee, J. H. M. and Stuckey, P. J.: Finite Domain Bounds Consistency Revisited, in *Proceedings of the 19th Australian Joint Conference on Artificial Intelligence*, LNCS 4304, Springer-Verlag, 2006, pp. 49–58.

[9] Codish, M., Fekete, Y., Fuhs, C. and Schneider-Kamp, P.: Optimal Base Encodings for Pseudo-Boolean Constraints, in *Proceedings of the 17th International Conference on Tools and Algorithms for Construction and Analysis of Systems* (*TACAS 2011*), LNCS 6605, Springer-Verlag, 2011, pp. 189–204.

[10] Crawford, J. M. and Baker, A. B.: Experimental Results on the Application of Satisfiability Algorithms to Scheduling Problems, in *Proceedings of the 12th National Conference on Artificial Intelligence* (*AAAI 1994*), Vol. 2, AAAI Press / The MIT Press, 1994, pp. 1092–1097.

[11] Darwiche, A. and Pipatsrisawat, K.: Complete Algorithms, *Handbook of Satisfiability*, ISO Press, 2009, pp. 99–130.

[12] Davis, M., Logemann, G. and Loveland, D. W.: A Machine Program for Theorem-Proving, *Communications of the ACM*, Vol. 5, No. 7(1962), pp. 394–397.

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

[14] de Moura, L. M. and Bjørner, N.: Z3: An Efficient SMT Solver, in *Proceedings of the 14th International Conference on Tools and Algorithms for Construction and Analysis of Systems* (*TACAS 2008*), LNCS 4963, Springer-Verlag, 2008, pp. 337–340.

[15] Drechsler, R., Junttila, T. A. and Niemelä, I.: Non-Clausal SAT and ATPG, *Handbook of Satisfiability*, ISO Press, 2009, pp. 655–693.

[16] Eén, N. and Sörensson, N.: An Extensible SAT-solver, in *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing* (*SAT 2003*), LNCS 2919, Springer-Verlag, 2003, pp. 502–518.

[17] Eén, N. and Sörensson, N.: Translating Pseudo-Boolean Constraints into SAT, *Journal on Satisfiability, Boolean Modeling and Computation*, Vol. 2, No. 1–4(2006), pp. 1–26.

[18] Gelder, A. V.: Another Look at Graph Coloring via Propositional Satisfiability, *Discrete Applied Mathematics*, Vol. 156, No. 2(2008), pp. 230–243.

[19] Gent, I. P.: Arc Consistency in SAT, in *Proceedings of the 15th European Conference on Artificial Intelligence* (*ECAI 2002*), 2002, pp. 121–125.

[20] Gomes, C. P., Selman, B. and Kautz, H. A.: Boosting Combinatorial Search Through Randomization, in *Proceedings of the 15th National Conference on Artificial Intelligence* (*AAAI 1998*), 1998, pp. 431–437.

[21] Hebrard, E.: Mistral, a Constraint Satisfaction Library, in *Proceedings of the 3rd International CSP Solver Competition*, 2008, pp. 31–39.

[22] Huang, J.: Universal Booleanization of Constraint Models, in *Proceedings of the 14th International Joint Conference on Principles and Practice of Constraint Programming* (*CP 2008*), LNCS 5202, Springer-Verlag, 2008, pp. 144–158.

[23] Inoue, K., Soh, T., Ueda, S., Sasaura, Y., Banbara, M. and Tamura, N.: A Competitive and Cooperative Approach to Propositional Satisfiability, *Discrete Applied Mathematics*, Vol. 154, No. 16(2006), pp. 2291–2306.

[24] 井上克巳, 田村直之: 特集「最近の SAT 技術の発展」, 人工知能学会誌 Vol. 25, No. 1 (2010), pp. 56–129.

[25] Iwama, K. and Miyazaki, S.: SAT-Variable Complexity of Hard Combinatorial Problems, in *Proceedings of the IFIP 13th World Computer Congress*, Vol. 1, 1994, pp. 253–258.

[26] 岩沼宏治, 鍋島英知: SMT: 個別理論を取り扱う SAT 技術, 人工知能学会誌, Vol. 25, No. 1(2010), pp. 86–95.

[27] Kasif, S.: On the Parallel Complexity of Discrete Relaxation in Constraint Satisfaction Networks, *Artificial Intelligence*, Vol. 45, No. 3(1990), pp. 275–286.

[28] Kautz, H., Selman, B. and Hoffmann, J.: SatPlan: Planning as Satisfiability, in *Proceedings of the 5th International Planning Competition* (*IPC-5*), 2006.

[29] Kroening, D.: Software Verification, *Handbook of Satisfiability*, ISO Press, 2009, pp. 505–532.

[30] Le Berre, D. and Lynce, I.: CSP2SAT4J: A Simple CSP to SAT translator, in *Proceedings of the second CSP Competition*, 2008, pp. 43–54.

[31] Marques-Silva, J. P. and Sakallah, K. A.: GRASP: A Search Algorithm for Propositional Satisfiability, *IEEE Transactions on Computers*, Vol. 48, No. 5(1999), pp. 506–521.

[32] Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L. and Malik, S.: Chaff: Engineering an Efficient SAT Solver, in *Proceedings of the 38th Design Automation Conference* (*DAC 2001*), 2001, pp. 530–535.

[33] 鍋島英知, 岩沼宏治, 井上克巳: GlueMiniSat2.2.5: 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, 日本ソフトウェア科学会第 28 回大会 講演論文集, 2011.
[34] 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(2010), pp. 68–76.
[35] Nabeshima, H., Soh, T., Inoue, K. and Iwanuma, K.: Lemma Reusing for SAT based Planning and Scheduling, in *Proceedings of the International Conference on Automated Planning and Scheduling 2006* (*ICAPS 2006*), 2006, pp. 103–112.
[36] Ohmura, K. and Ueda, K.: c-sat: A Parallel SAT Solver for Clusters, in *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing* (*SAT 2009*), LNCS 5584, Springer-Verlag, 2009, pp. 524–537.
[37] Petke, J. and Jeavons, P.: The Order Encoding: From Tractable CSP to Tractable SAT, in *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing* (*SAT 2011*), LNCS 6695, Springer-Verlag, 2011, pp. 371–372.
[38] Rintanen, J.: Planning and SAT, *Handbook of Satisfiability*, IOS Press, 2009, pp. 483–504.
[39] Rossi, F., van Beek, P. and Walsh, T.: *Handbook of Constraint Programming*, Elsevier Science Inc., New York, NY, USA, 2006.
[40] Roussel, O. and Manquinho, V. M.: Pseudo-Boolean and Cardinality Constraints, *Handbook of Satisfiability*, IOS Press, 2009, pp. 695–733.
[41] Silva, J. P. M., Lynce, I. and Malik, S.: Conflict-Driven Clause Learning SAT Solvers, *Handbook of Satisfiability*, ISO Press, 2009, pp. 131–153.
[42] Tadesse, D., Sheffield, D., Lenge, E., Bahar, R. I. and Grodstein, J.: Accurate Timing Analysis using SAT and Pattern-dependent Delay Models, in *Proceedings of the conference on Design, automation and test in Europe*, DATE '07, San Jose, CA, USA, EDA Consortium, 2007, pp. 1018–1023.
[43] Tamura, N., Taga, A., Kitagawa, S. and Banbara, M.: Compiling Finite Linear CSP into SAT, in *Proceedings of the 12th International Joint Conference on Principles and Practice of Constraint Programming* (*CP 2006*), LNCS 4204, Springer-Verlag, 2006, pp. 590–603.
[44] Tamura, N., Taga, A., Kitagawa, S. and Banbara, M.: Compiling Finite Linear CSP into SAT, *Constraints*, Vol. 14, No. 2(2009), pp. 254–272.
[45] 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(2010), pp. 77–85.
[46] Tanjo, T., Tamura, N. and Banbara, M.: Towards a Compact and Efficient SAT-Encoding of Finite Linear CSP, in *Proceedings of the 9th International Workshop on Modelling and Reformulating Constraint Satisfaction Problems* (*ModRef 2010*), 2010.
[47] Tanjo, T., Tamura, N. and Banbara, M.: A Compact and Efficient SAT-Encoding of Finite Domain CSP, in *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing* (*SAT 2011*), LNCS 6695, Springer-Verlag, 2011, pp. 375–376.
[48] Tanjo, T., Tamura, N. and Banbara, M.: Proposal of a Compact and Efficient SAT Encoding using a Numeral System of Any Base, in *Proceedings of the 1st International Workshop on the Cross-Fertilization Between CSP and SAT* (*CSP-SAT 2011*), 2011.
[49] Tanjo, T., Tamura, N. and Banbara, M.: Azucar: A SAT-Based CSP Solver using Compact Order Encoding (Tool Presentation), in *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing* (*SAT 2012*), LNCS 7317, Springer-Verlag, 2012, pp. 456–462.
[50] The choco team: choco: an Open Source Java Constraint Programming Library, in *Proceedings of the 3rd International CSP Solver Competition*, 2008, pp. 7–13.
[51] 梅村晃広: SAT ソルバ・SMT ソルバの技術と応用, コンピュータソフトウェア, Vol. 27, No. 3(2010), pp. 24–35.
[52] Walsh, T.: SAT v CSP, in *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming* (*CP 2000*), 2000, pp. 441–456.
[53] Zankl, H. and Middeldorp, A.: Satisfiability of Non-linear (Ir)rational Arithmetic, in *Proceedings of the 17th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning* (*LPAR-17*), LNCS 6397, Springer-Verlag, 2010, pp. 481–500.

丹生智也

2007 年神戸大学工学部情報知能工学科卒業．2009 年同大学大学院工学研究科博士課程前期課程情報知能学専攻修了．2012 年同研究科博士課程後期課程情報知能学専攻修了．2012 年より新領域融合研究センター 融合プロジェクト特任研究員．博士 (工学)．制約プログラミング，論理プログラミング，SAT 技術，解集合プログラミング，擬似ブール制約等に興味をもつ．日本ソフトウェア科学会会員，人工知能学会会員．

田村直之

1980 年神戸大学理学部物理学科卒業．1985 年同大学大学院自然科学研究科博士課程システム科学専攻修了．学術博士．1985 年日本 IBM 東京基礎研究所入社．1988 年神戸大学工学部勤務．2003 年より神戸大学情報基盤センター教授．論理プログラミング，制約プログラミング，SAT 技術等に興味をもつ．日本ソフトウェア科学会，情報処理学会会員．

番原睦則

1994 年神戸大学理学部数学科卒業．1996 年同大学大学院自然科学研究科博士課程前期数学専攻修了．1996 年国立奈良工業高等専門学校助手．1998 年同校講師．2003 年より神戸大学情報基盤センター講師．2007 年同校准教授．博士 (工学)．論理プログラミング，制約プログラミング，SAT 技術等に興味をもつ．日本ソフトウェア科学会，情報処理学会会員．