命題論理の充足可能性判定問題への符号化を用いた制約充足問題の解法に関する研究

丹生，智也

(Degree)
博士（工学）
(Date of Degree)
2012-03-25
(Date of Publication)
2014-01-15
(Resource Type)
doctoral thesis
(Report Number)
甲5490
(URL)
https://hdl.handle.net/20.500.14094/D1005490

博　士　論　文

命題論理の充足可能性判定問題への符号化を用いた制約充足問題の解法に関する研究

平成24年1月

神戸大学大学院工学研究科

丹生 智也

要旨

制約充足問題 (Constraint Satisfaction Problems; **CSP**) は，変数の有限集合と各変数が取りうる値の集合 (ドメイン)，満たすべき条件が与えられたとき，条件を満たす変数への値割り当てを探索する問題である．また制約を満たす解のうち最適なものを求める問題を**制約最適化問題**(Constraint Optimization Problem; **COP**) という．スケジューリング問題やグラフ彩色問題をはじめとする多くの組み合わせ問題は CSP または COP として定式化できる．また 2005 年から国際 CSP ソルバー競技会が開催されており，**制約ソルバー**の高速化に関する研究が活発に行われている．

また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り，Cook により最初に NP 完全性が示された問題である．近年，監視リテラルや矛盾からの節学習をはじめとする SAT 求解の高速化技術が発展したことで，DPLL 手続きに基づく SAT ソルバーにより，実問題から得られる構造的な SAT 問題を高速に求解することが可能となった．これに伴い，有界モデル検査，プランニング，スケジューリング，自動テストパターン生成，ソフトウェア検証等の分野においては，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**SAT 型システム**が成功を収めている．特に，CSP や COP を解く SAT 型システムは**SAT 型制約ソルバー**と呼ばれ，活発に研究が行われている．

SAT 型制約ソルバーでは，SAT ソルバーの性能だけでなく，SAT への符号化法も同様に重要であることが知られている．CSP から SAT への符号化は直接符号化，支持符号化，対数符号化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，d を各変数のドメインサイズとすると，直接符号化と支持符号化では各 3 項制約が $O(d^3)$ 個の節に符号化されるため，ドメインサイズが 10^2 未満の小規模の問題にしか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での範囲伝播しか実現できず，矛盾の検出のために桁数 $\log_2 d$ に比例する変数の値を決定する必要があるため，効率が悪いという問題がある．

本論文では DPLL 手続きに基づく系統的 SAT ソルバーを用いることを前提とし，既存の SAT 符号化の問題点を解決する以下の 2 つの SAT 符号化の提案を行う．

- **順序符号化**
  この符号化は，Crawford と Baker がジョブショップ・スケジューリング問題に適用した方法を CSP に適用可能なように一般化したものである．SAT ソ

ルバーでの単位伝播がCSPソルバーの範囲伝播に対応しており，直接符号化，支持符号化，対数符号化等の他の符号化と比較して，効率の良い求解が可能である．また，各3項制約は$O(d^2)$個の節に符号化されるため，ドメインサイズが10^3以下の小規模および中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**
  ドメインサイズが10^2以下から10^7までの広い範囲の問題へ適用可能かつ効率的であることを目指して設計した方法である．コンパクト順序符号化の基本アイデアは以下の2つである．

  - 各整数変数をB進法を用いて表す$(B \geq 2)$．すなわち，各整数変数xを$m = \lceil \log_B d \rceil$桁の変数で表現する．
  - 各桁を順序符号化を用いてSATに符号化する．

  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．コンパクト順序符号化は，SATソルバーの単位伝播が最上位桁での範囲伝播に対応しているため，一般に対数符号化よりも効率が良い．また各整数を$m = \lceil \log_B d \rceil$桁で表すと，3項制約$z = x + y$と$z = xy$はそれぞれ$O(mB^2)$個と$O(mB^3 + m^2B^2)$個の節に符号化され，順序符号化よりもはるかに少なくなるため，ドメインサイズが10^7程度の大規模な問題に対しても適用可能である．

グラフ彩色問題による実験結果，順序符号化を採用したSAT型制約ソルバーSugarの2008年度の国際制約ソルバー競技会の結果，釣合い型不完備ブロック計画構成問題による実験結果等により，最先端の制約ソルバーと比較しても順序符号化を採用したSAT型制約ソルバーが有効であることを示した．

また，コンパクト順序符号化の有効性を検証するため，様々なドメインサイズの問題による性能評価を行い，以下の結果が得られた．

- コンパクト順序符号化が，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対して適用可能であることを確認した．

- 順序符号化や既存のSAT符号化，最先端の制約ソルバーであるMistralとchocoがほとんど解くことのできない大規模な問題であっても，コンパクト順序符号化は平均的に高速な求解を行えることを確認した．

最後に，今回提案した順序符号化およびコンパクト順序符号化は既存のSAT符号化の欠点を解消したものであり，これによりSAT型システムの適用範囲をさらに広げることができると考えている．

目 次

第1章　序論

制約充足問題 (Constraint Satisfaction Problems; **CSP**)[56] は，変数の有限集合と各変数が取りうる値の集合 (ドメイン)，満たすべき条件が与えられたとき，条件を満たす変数への値割り当てを探索する問題である．また制約を満たす解のうち最適なものを求める問題を**制約最適化問題**(Constraint Optimization Problem; **COP**) という．スケジューリング問題やグラフ彩色問題をはじめとする多くの組み合わせ問題はCSP またはCOP として定式化できる．また2005 年から国際CSP ソルバー競技会 [1]が開催されており，**制約ソルバー**の高速化に関する研究が活発に行われている．

また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り，最初に NP 完全性が示された問題である [8, 35, 73, 14]．

近年のSAT 求解のための高速化技術の発展により，DPLL 手続きに基づくSAT ソルバーが，実問題から得られる構造的なSAT 問題を高速に求解することが可能となった．これに伴い，有界モデル検査 [7, 6]，プランニング [38, 55]，スケジューリング [15, 64]，自動テストパターン生成 [20]，ソフトウェア検証 [39] 等の個々の分野においては，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**SAT 型システム**が成功を収めている [8]．特に，CSP やCOP を解くSAT 型システムは**SAT 型制約ソルバー**と呼ばれ，活発に研究が行われている．

SAT 型制約ソルバーでは，SAT ソルバーの性能だけでなく，SAT への符号化法も同様に重要であることが知られている [54]．CSP からSAT への符号化は直接符号化 [18, 77, 66]，支持符号化 [37, 27, 66]，対数符号化 [36, 25, 66] 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．例えば変数のドメインサイズを d とすると，直接符号化と支持符号化は各3 項制約を $O(d^3)$ 個の節に符号化するため，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，効率が悪いという問題がある．

[1] http://www.cril.univ-artois.fr/CSC09/

本論文では DPLL 手続きに基づく系統的SAT ソルバーを用いることを前提とし，既存の SAT 符号化の問題点を解決する以下の 2 つのSAT 符号化の提案を行う．

- **順序符号化**[63, 64, 66]
  これは，Crawford と Baker がジョブショップ・スケジューリング問題に適用した方法を CSP に適用可能なように一般化したものである．SAT ソルバーでの単位伝播が制約ソルバーの範囲伝播に対応しており，直接符号化，支持符号化，対数符号化等の他の符号化と比較して，効率の良い求解が可能である．また，各 3 項制約は $O(d^2)$ 個の節に符号化されるため，ドメインサイズが 10^3 以下の小規模および中規模な問題に対して適用可能である．

- **コンパクト順序符号化**[69, 70, 68, 71]
  この符号化の基本アイデアは以下の 2 つである．

  - 各整数変数を B 進法を用いて表す ($B \geq 2$)．すなわち，各整数変数 x は $m = \lceil \log_B d \rceil$ 桁の変数で表現される．
  - 各桁を順序符号化を用いて SAT に符号化する．

  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．コンパクト順序符号化は，SAT ソルバーの単位伝播が最上位桁での範囲伝播に対応しているため，一般に対数符号化よりも効率が良い．また各整数を $m = \lceil \log_B d \rceil$ 桁で表すと，3 項制約 $z = x+y$ と $z = xy$ はそれぞれ $O(mB^2)$ 個と $O(mB^3+m^2B^2)$ 個の節に符号化され，順序符号化よりもはるかに少なくなるため，ドメインサイズが 10^7 程度の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は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かつ効率的であることを目指して設計された方法である．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下で，順序符号化および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 章で SAT および SAT ソルバーについて述べる．第 3 章で制約充足問題，制約最適化問題および制約ソルバーについて述べた後，第 4 章で SAT 型制約ソルバーと既存の SAT 符号化である直接符号化，支持符号化，対数符号化についても述べる．第 5 章で小規模および中規模な問題へ適用可能な新しい符号化である順序符号化について述べた後，第 6 章でグラフ

彩色問題，順序符号化を採用したSAT型制約ソルバー Sugar の2008年度の制約ソルバー競技会の結果および釣合い型不完備ブロック計画 (Balanced Incomplete Block Design; ***BIBD***) 構成問題 を用いた性能評価の結果を示す．第7章では大規模な問題へ適用可能な新しいSAT符号化である コンパクト順序符号化について述べる．第8章で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の結果を示す．最後に，第9章で結論を述べる．

第2章　SATとSATソルバー

本章では，命題論理式の充足可能性判定問題 (SAT) と，SAT を解くシステムである SAT ソルバーについて述べる．まず任意の命題論理式は，ド・モルガン律や Tseitin 変換等を用いることで，SAT の標準的な入力である連言標準形へと変換できることを示す．また，近代的な SAT ソルバーの多くが採用している DPLL 手続きについて述べる．

2.1　SAT の定義

SAT は，与えられた命題論理式を真にするような命題変数への真理値割り当て (truth assignment) が存在するか否かを判定する問題である．真にする真理値割り当てが存在すれば元の命題論理式は**充足可能** (satisfiable) と呼ばれ，存在しなければ**充足不能** (unsatisfiable) と呼ばれる．与えられた SAT 問題 (SAT problem instance) が充足可能な場合，その問題を真にする真理値割り当てが**解**となる．

通常，命題論理式は**連言標準形** (Conjunctive Normal Form; CNF) で与えられる．すなわち，全体の論理式はいくつかの**節** (clause) の連言 (AND) であり，各節はいくつかの**リテラル** (literal) の選言 (OR)，各リテラルは**命題変数** (propositional variable) あるいはその否定である．

2.2　Tseitin 変換

任意の命題論理式は，ド・モルガン律等を用いて，否定が命題変数の直前のみに現れる**否定標準形**(Negative Normal Form; NNF) に変換できる．否定標準形の命題論理式は，分配律等を用いて連言標準形に変換できるが，得られる論理式の長さが元の論理式の長さの指数関数的になる可能性がある．

Tseitin は，元の論理式の長さと線形の長さの連言標準形に変換する **Tseitin 変換** (Tseitin transformation) の方法を示した [54]．Tseitin 変換では，$P \vee (Q \wedge R)$ 等について，新しい命題変数 p を導入し，$Q \wedge R$ の部分を p で置き換えるとともに，$p \Leftrightarrow (Q \wedge R)$ と同値な論理式として $(\neg p \vee Q) \wedge (\neg p \vee R) \wedge (p \vee \neg Q \vee \neg R)$ を追加する．

ただし，元の論理式が否定標準形であれば，以下で示すように $(\neg p \vee Q) \wedge (\neg p \vee R)$ のみの追加で良い．

Tseitin 変換を行う前の論理式と変換後の論理式は，同値ではないが，両者の充足可能性が一致する**充足同値** (equisatisfiable) になっている．

定理 1 (Tseitin 変換)．A を否定標準形の命題論理式，B を命題論理式とし，命題変数 p は B 中に現れておらず A 中には $\neg p$ の形で現れていないとする．$A[B]$ により，A 中のすべての p の出現を B で置き換えた論理式を表す時，$A[B]$ と $A \wedge (\neg p \vee B)$ は充足同値であり，以下が成り立つ．

$$A[B] \text{ が充足可能} \iff A \wedge (\neg p \vee B) \text{ が充足可能}$$

Proof. ($\Longrightarrow$) $A[B]$ を充足する真理値割り当てを α とする．$A[B]$ 中に p が現れないことから $\alpha'(p) = \alpha(B)$，$\alpha'(q) = \alpha(q)$ ($q \neq p$ の時) により，α' を定める．明らかに $\alpha'(A) = \alpha(A[B]) = 1$ であり，$\alpha'(p) = \alpha(B) = \alpha'(B)$ より $\alpha'(\neg p \vee B) = 1$ となり，α' は $A \wedge (\neg p \vee B)$ を充足する．
($\Longleftarrow$) $A \wedge (\neg p \vee B)$ を充足する真理値割り当てを α' とすると，$\alpha'(A) = 1$ かつ $\alpha'(\neg p \vee B) = 1$ である．α' の定義域から p を削除したもので α を定める．この時，$\neg p$ が現れていない任意の否定標準形の論理式 C について，$\alpha'(C) = 1$ ならば $\alpha(C[B]) = 1$ であることを，C に関する構造的帰納法で証明する．

C が命題変数 p の場合，$\alpha(C[B]) = \alpha(B) = \alpha'(B)$ だが，$\alpha'(p) = 1$ と $\alpha'(\neg p \vee B) = 1$ より $\alpha'(B) = 1$ である．

C が命題変数 q あるいは $\neg q$ (ただし q は p と異なる) の場合，$\alpha(C[B]) = \alpha(C) = \alpha'(C) = 1$ である．

C が $C_1 \wedge C_2$ の場合，$\alpha'(C) = 1$ より $\alpha'(C_1) = 1$ かつ $\alpha'(C_2) = 1$ である．帰納法の仮定より $\alpha(C_1[B]) = 1$ かつ $\alpha(C_2[B]) = 1$，したがって $\alpha(C[B]) = 1$ である．C が $C_1 \vee C_2$ の場合，$C_1 \wedge C_2$ の場合と同様にして $\alpha(C[B]) = 1$ が示される．
よって，α は $A[B]$ を充足する． □

例えば，否定標準形の論理式 $P \vee (Q \wedge R)$ について Tseitin 変換を行う場合，新しい命題変数 p を準備し，A を $P \vee p$，B を $Q \wedge R$ とする．上の定理より $P \vee (Q \wedge R)$ と $(P \vee p) \wedge (\neg p \vee (Q \wedge R))$ すなわち $(P \vee p) \wedge (\neg p \vee Q) \wedge (\neg p \vee R)$ は充足同値である．このような変換を同様の部分論理式に対して繰り返し行えば，最終的に連言標準形の論理式が得られる．

2.3 SAT ソルバー

SAT の解を探索するプログラムは，**SAT ソルバー** (SAT solver) と呼ばれる．SAT ソルバーには，系統的に解を探索し，充足可能または充足不能を判定する**系**

統的 SAT ソルバーと，確率的に解を探索することで，充足可能のみを判定する**確率的 SAT ソルバー**が存在する．

近年 SAT ソルバーの高速化技術の発展により，特に実問題から得られる構造的な SAT 問題に対して，**DPLL 手続き** [17] に基づく系統的 SAT ソルバーの性能が著しく向上している [16, 58, 46, 45, 48]．

また近代的な SAT ソルバーは DPLL 手続き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 [61]．

- 矛盾からの節学習 (Conflict Driven Clause Learning; CDCL) [58, 41]
- 変数選択ヒューリスティックス [44]
- 非時間順バックトラック [41]
- 監視リテラル [44]
- リスタート戦略 [29]

本論文では DPLL 手続き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 ソルバーと呼び，DPLL 手続きを用いて効率的に求解が可能な SAT 符号化について述べる．

2.3.1 DPLL 手続き

この節では，以下の例を用いて DPLL 手続きの説明を行う．

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

DPLL 手続きの初期状態では，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** (Unit Propagation) という．単位節がなく

なるまで，DPLL 手続き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 手続き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 [44]．また決定変数を数多く選択すると，探索空間が大きくなってしまい実行速度が低下すると考えられる．このため DPLL 手続きに基づく 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．

第3章　制約充足問題

制約充足問題 (Constraint Satisfaction Problem; CSP) は，各変数 (variable) に与えられたドメイン (domain) から値を割り当てることで，与えられた制約 (constraint) の全てを満たすことができるかどうかを判定する問題である [2, 9]．すべての制約を満たす値割り当て (assignment) が存在する場合，元の制約充足問題は**充足可能** (satisfiable) であるといい，その値割り当てが**解**となる．値割り当てが存在しない場合，元の制約充足問題は**充足不能** (unsatisfiable) であるという．

制約充足問題は，一般には実数や集合など様々なドメイン上で展開されるが，本論文では実用上多くの応用を含む整数の有限領域 (finite domain) 上の制約充足問題を対象とする．

整数有限領域上の制約充足問題は，形式的には以下のように定義できる．

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, D, P, C) である．

- X は，整数変数の有限集合である．
- D は X から $\mathbb{Z}$ の有限部分集合全体への写像であり，各変数の取りうる値集合 (**ドメイン**) 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．

本論文では説明の簡単化のため，制約 C を $c_1 \wedge c_2 \wedge \ldots \wedge c_n$ のように，制約の連言として表すことがある．

制約には，算術論理演算等で条件が記述される内延的制約，制約を満足する (あるいは違反する) 値の組の集合が与えられる外延的制約， alldifferent 等に代表されるグローバル制約からなる．

内延的制約 (intentional constraint) は，通常の算術演算および算術比較に加え，否定 ($\neg$)，連言 ($\wedge$)，選言 ($\vee$)，含意 ($\Rightarrow$)，同値 ($\Leftrightarrow$) 等の論理演算を用いて条件を記述したものである．例えば，$(x_1 + 2 \leq x_2) \vee (x_2 + 3 \leq x_3)$ は内延的制約である．

外延的制約 (extensional constraint) では，制約を満足する値の組の集合である支持点集合 (set of supports)，あるいは制約を違反する値の組の集合である違反点集合 (set of conflicts) が与えられる．例えば，支持点集合 $R = \{(0,0),(1,1),(2,2)\}$ とし整数変数 x_1, x_2 のドメインを $\{0,1,2\}$ とする．この時，外延的制約 $R(x_1, x_2)$ は $x_1 = x_2$ を表す．同時に，$R = \{(0,0),(1,1),(2,2)\}$ が違反点集合の場合，外延的制約 $\bar{R}(x_1, x_2)$ は $x_1 \neq x_2$ を表す ($\bar{R}$ は R の補集合を表す)．

グローバル制約 (global constraint) は，複数の変数に対する複雑な (しかし意味のある) 条件を簡潔に表すために導入された．例えば alldifferent$(x_1, x_2, \ldots, x_n)$ は，x_i が互いに異なることを表す．$x_i \neq x_j$ を個別に記述するよりも簡潔であり，また効率的な求解アルゴリズムが知られている [28]．このようなグローバル制約は，記述性および効率性の向上を目的として制約ソルバーや制約プログラミングシステムに数多く取り入れられている．これらのグローバル制約を集約した Global Constraint Catalog[1] には，350 以上のグローバル制約が紹介されている．

CSP (X, D, P, C) への値割り当ては，組 (α, β) で表される．ここで，α は X から $\mathbb{N}$ への写像であり，β は P から $\{1,0\}$ への写像である (1，0 はそれぞれ真と偽を表す)．また論理式 C を充足し，すべての $x \in X$ に対して $\alpha(x) \in D(x)$ であるような値割り当て (α, β) が存在するとき，CSP は**充足可能**であるといい，その時の値割り当てをその CSP の**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．

3.1 制約最適化問題

単純に条件を満たす解を探索するだけでなく，制約を満たす解のうち最適なものを求める問題を**制約最適化問題** (Constraint Optimization Problem; COP) という．制約最適化問題では，条件を満たす解のうち，指定された目的関数 (objective function) あるいは目的変数 (objective variable) の値を最小 (あるいは最大) にする解を求めることが目的である．

整数有限領域上の制約最適化問題は，形式的には以下のように定義できる．

定義 2 (制約最適化問題)．COP は以下を満たす組 (X, D, P, C, v) である．

- (X, D, P, C) は CSP である．
- 変数 $v \in X$ は最小化すべき目的変数を表す [2]．

[1] `http://www.emn.fr/z-info/sdemasse/gccat/`

[2] COP を最小化問題とみなしても一般性を失わない．

8	1	6
3	5	7
4	9	2

x_1	x_2	x_3
x_4	x_5	x_6
x_7	x_8	x_9

図 3.1: 3×3 の魔方陣

3.2 CSP の例

ここでは，CSP の例として魔方陣とグラフ彩色問題を取り上げる．その他の問題例については，XCSP [3] や CSPLib [4] の Web サイトなどを参照されたい．

3×3 の魔方陣は，1 から 9 の数字を 3 行 3 列のマスに配置し，各行，各列および二つの対角線について配置されている 3 つの数字の和がいずれも 15 となるようにする問題である．図 3.1 の左は解の一つを表している．

これを CSP として定式化するために，図 3.1 の右側のように整数変数を各マスに割り当て，各変数のドメイン $D(x_i)$ を $\{1, 2, \ldots, 9\}$ と定める．

必要な制約は，各マスの数字が互いに異なることを表す $\mathrm{alldifferent}(x_1, x_2, \ldots, x_9)$，および各行，各列および 2 つの対角線の和が 15 に等しいことを表す $x_1+x_2+x_3 = 15$ 等である．

次の例として，最適化コンパイラのレジスタ割り付け，無線の周波数割り当て等の応用があるグラフ彩色問題を取り上げる．有名なパズルである数独もグラフ彩色問題として定式化できる．

グラフ彩色問題 (Graph Coloring Problem; GCP) は，与えられた有限無向グラフ G について，隣接する頂点が同色にならないように各頂点を彩色する時，必要となる最小の色数を求める問題である．最小の色数は**彩色数** (chromatic number) と呼ばれ，$\chi(G)$ で表される．

彩色数を求める問題は NP 困難であることが知られている．また，与えられた自然数 $k \geq 3$ について，グラフが k 色以下で彩色可能かどうかを決定する問題は NP 完全である．グラフが k 色以下で彩色可能な時，そのグラフは **k-彩色可能** (k-colorable) であるという．

例えば，図 3.2 に示されているグラフは，3-彩色可能であるが 2-彩色可能ではない．したがって，このグラフの彩色数は 3 である．

グラフが k-彩色可能かどうかを決定する問題は，制約充足問題として定式化できる．グラフの各頂点に対して整数変数 x_i を割り当て，各整数変数のドメイン $D(x_i)$ を $\{0, 1, \ldots, k-1\}$ と定める．また，すべての辺について，対応する整数変数の値が異なることを意味する $x_i \neq x_j$ を制約として付け加える．

[3] http://www.cril.univ-artois.fr/lecoutre/benchmarks.html
[4] http://www.csplib.org

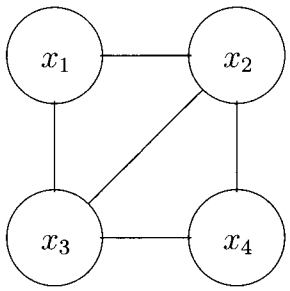

図 3.2: グラフ彩色問題の例

例えば，図 3.2 に示されているグラフが3-彩色可能かどうかを決定する問題は，CSP$(X, D, \emptyset, C)$ として以下のように定式化できる．

$$\begin{aligned} X &= \{x_1, x_2, x_3, x_4\} \\ D(x_i) &= \{0, 1, 2\} \qquad (i = 1, 2, 3, 4) \\ C &= \{x_1 \neq x_2,\ x_1 \neq x_3, \\ &\quad\ x_2 \neq x_3,\ x_2 \neq x_4,\ x_3 \neq x_4\} \end{aligned}$$

彩色数を求める問題を制約最適化問題として定式化する場合，事前に彩色数の上界を求めておく必要がある．彩色数はグラフの最大次数+1 以下であることが知られているのでこれを用いるか，貪欲法等の単純なアルゴリズムで上界を求めれば良い．

3.3 制約ソルバー

CSP の解を探索するプログラムは，**制約ソルバー** (constraint solver) と呼ばれる．また，プログラミング言語の一部に (あるいはライブラリとして) 制約ソルバーが組込まれ，プログラミング言語とより融合したシステムは**制約プログラミング** (constraint programming) と呼ばれる．

これらの制約ソルバーや制約プログラミングシステムには，clp (FD) [12]，ILOG Solver [5]，choco [72]，Cream [6] などがある．これらのシステムは，制約に違反する値を削除する**整合性アルゴリズム** (consistency algorithm) を用いフォワード・チェック法やアーク整合性維持法といった制約伝播 (constraint propagation) アルゴリズムにより探索空間を削減する工夫を行っている．また，単純な**バックトラック法** (backtracking) による探索だけでなく，矛盾の原因になった値割り当てに後

[5] http://www.ilog.com/products/cp/
[6] http://bach.istc.kobe-u.ac.jp/cream/

戻りするバックジャンプ法 (backjumping) などを用いて効率的な探索を実現している．本節では，代表的な整合性アルゴリズムであるフォワード・チェック法とアーク整合性維持法について述べる．その他のアルゴリズムの詳細は本稿の範囲を越えるため，文献 [2, 9] 等を参照されたい．

制約プログラミングシステムでは，ベースとなっている Prolog，C++，Java などの言語の構文を用いて CSP を記述するが，CSP を記述するための**制約モデリング言語**(constraint modeling language) を用いる場合もある．OPL [75]，Zinc [19]，XCSP などは制約モデリング言語の例である．特に，XCSP は国際 CSP ソルバー競技会で使用され，多数のベンチマーク問題が公開されている．

3.3.1 整合性アルゴリズム

(一般) アーク整合性 ((Generalized) Arc Consistency; (G)AC) は非常によく知られた考えである [56]．本論文では (G)AC を以下のように定義する．

定義 3 ((一般) アーク整合性 ((G)AC))．組 (X, D, P, C) を CSP とし，$x \in X$ を整数変数，$c \in C$ を x を含む制約とする．

- 制約 c を真にし，$\alpha(x) = v$ となる値割り当て (α, β) が存在する時かつその時に限り，**値 $v \in D(x)$ は c と整合しているという**．
- $D(x)$ のすべての値が c と整合している時かつその時に限り，**変数 x は c 上でアーク整合であるという**．

CSP (X, D, P, C)，$p \in P$ と p を含む制約 $c \in C$ に対しても同様に，アーク整合性が定義できる．以下では説明の簡単化のため，$P = \emptyset$ の場合の整合性アルゴリズムについて述べる．

変数 x を制約 c 上でアーク整合にする手続き revise を図 3.3 に示す．revise では，x のドメインの各値 v について，c を真にし，$\alpha(x) = v$ となる値割り当てが存在するかどうかを確認し，そのような値割り当てがない場合には，x のドメインから v を削除する．この手続きは，x のドメインを削除した場合には true を，それ以外の場合は false を返す．

フォワード・チェック法

フォワード・チェック法 (Forward Checking; **FC**) は，最近に具体化された変数とまだ具体化されていない変数間についてのみアーク整合性アルゴリズムを適用する方法である [77]．

```
procedure revise(x, c);
begin
 CHANGE := false;
  foreach v ∈ D(x) do
    if c を真にし α(x) = v となる値割り当て (α, β) が存在しな
い then
      remove v from D(x);
      CHANGE := true;
    end if
  end foreach
 return CHANGE;
end
```

図 3.3: アーク整合性アルゴリズム

```
procedure FC(CSP, i);
begin
  foreach c ∈ C do
    foreach j := i+1 to n do
      if c が x_i と x_j を含む then
        revise(x_j, c)
      end if
    end foreach
  end foreach
end
```

図 3.4: フォワード・チェック法

図 3.4 に，$CSP(X, D, P, C)$ にフォワード・チェック法を適用する FC 手続きを示す (ただし，$X = \{x_1, x_2, \ldots, x_n\}$)．引数の i は最も最近具体化した変数の添字を表す．すなわち，$x_1, x_2, \ldots, x_{i-1}$ は既に具体化された変数であり，x_i は最も最近具体化した変数である．また，$x_1, x_2, \ldots, x_{i-1}$ に対しては，すでにフォワード・チェック法が適用されているとする．もし FC 手続きにより，いずれかの変数のドメインが空になった場合には，現在の値割り当てを拡張して解を得ることができないため，バックトラックを行う．

アーク整合維持法

アーク整合維持法 (Maintaining Arc Consistency; **MAC**) は，すべての変数間にアーク整合性アルゴリズムを適用する方法であり，フォワード・チェック法よりも強い制約伝播アルゴリズムとなっている [2].

```
procedure MAC(CSP, i);
begin
 FC(C, i);
  foreach j := i+1 to n do
    foreach k := i+1 to n do
      foreach c ∈ C do
        if k ≠ j かつ c が x_j と x_k を含む
         revise(x_j, c)
        end if
      end foreach
    end foreach
  end foreach
end
```

図 3.5: アーク整合維持法

図 3.5 に，$CSP(X, D, P, C)$ にアーク整合維持法を適用するアルゴリズムを示す．FC 手続きの場合と同様に，引数の i は最も最近具体化した変数の添字を表す．

手続き MAC は，まず CSP と i に対してフォワード・チェック法を実行し，その後未割り当ての変数 x_j と x_k $(j, k \in \{i+1, i+2, \ldots, n\},\ j \neq k)$ を含む制約に対して revise 手続きを呼び出す．MAC 手続きの結果，いずれかの変数のドメインが空になった場合にはバックトラックを行う．

3.4 CSP の規模

第 4.2 節，第 5 章，第 7 章で各 SAT 符号化の評価の目安として用いるため，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 章で述べたように，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 CSP 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．

第4章　制約充足問題のSAT符号化

4.1　SAT型制約ソルバー

SAT型制約ソルバー (SAT-based constraint solver) はCSPを解くためのSAT型システムである．すなわちCSPをSATに符号化し，高速なSATソルバーを用いて元のCSPの解を探索する (図4.1参照)．第1節で述べたように，SAT型制約ソルバーでは，SATソルバーの性能だけでなく，SATへの符号化法も同様に重要である．

4.2　既存のSAT符号化

以下では主なSAT符号化である直接符号化，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．また本章と第5章，第7章では，以下の点について各符号化の特徴を考察する．

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．また乗算制約を含まない小規模，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数について考察する．

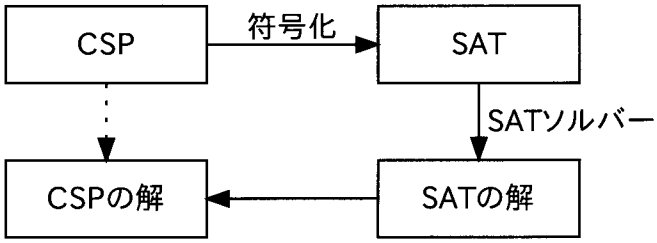

図 4.1: SAT 型制約ソルバー

4.2.1 直接符号化

直接符号化 (direct encoding) は最も広く用いられている SAT 符号化である．各整数変数 x と各整数定数 $a \in D(x)$ に対して，$x = a$ を表す命題変数 $p(x = a)$ を用いる [18, 77, 66].

整数変数 $x \in \{0..4\}$ に対して，命題変数 $p(x = 0), p(x = 1), p(x = 2), p(x = 3), p(x = 4)$ が用いられる．

x が 0 から 4 のうち少なくとも 1 つの値を取ることを表す **at-least-one** 節を以下のように表す．

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

また，x が 2 つ以上の値を同時に取らないことを表す **at-most-one** 節を以下のように表す [1].

$$\begin{array}{ll} \neg p(x = 0) \vee \neg p(x = 1) & \neg p(x = 0) \vee \neg p(x = 2) \\ \neg p(x = 0) \vee \neg p(x = 3) & \neg p(x = 0) \vee \neg p(x = 4) \\ \neg p(x = 1) \vee \neg p(x = 2) & \neg p(x = 1) \vee \neg p(x = 3) \\ \neg p(x = 1) \vee \neg p(x = 4) & \neg p(x = 2) \vee \neg p(x = 3) \\ \neg p(x = 2) \vee \neg p(x = 4) & \neg p(x = 3) \vee \neg p(x = 4) \end{array}$$

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．例えば，制約 $x \leq y$ の違反点 $(x = 3) \wedge (y = 2)$ を禁止する節は，$\neg p(x = 3) \vee \neg p(y = 2)$ と表現される (図 4.2 参照).

また，$x \geq 3$ と $y \leq 2$ は，以下の節に符号化される．

$$\begin{array}{lll} \neg p(x = 0) & \neg p(x = 1) & \neg p(x = 2) \\ \neg p(y = 3) & \neg p(y = 4) & \end{array}$$

[1] at-most-one 節をより少ない節で表す方法として，ラダー符号化 [26] やビットワイズ符号化 [53] が提案されている．

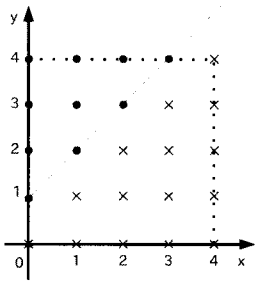

図 4.2: $x+1 \leq y$ の違反点

この場合には単位伝播が起きず，矛盾を検出できない．しかし，例えば $p(y=2)$ を真に割り当てれば，単位伝播により矛盾を検出できる．

符号化後の節数および伝播能力について

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 $O(d)$ 個の変数と $O(d^2)$ 個の節が必要となる．また，各 2 項制約は $O(d^2)$ 個の節に，各 3 項制約は $O(d^3)$ 個の節に符号化される．したがって，小規模ドメインサイズの問題に対してさえ約 $10^7 \sim 10^9$ 個の節が必要となってしまうため，中規模以上のドメインサイズの問題には実質的に適用できない．

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，制約ソルバーでのフォワード・チェック法に対応している [77]．例に挙げたような 2 項制約間の矛盾を検出するためには，最低でも決定変数が 1 つ必要となるため，あまり効率が良くない．

4.2.2 支持符号化

支持符号化 (support encoding) も直接符号化と同様に，各整数変数 x と各整数定数 $a \in D(x)$ に対して，$x=a$ を表す命題変数 $p(x=a)$ を用いる [37, 27]．各整数変数に対して，at-least-one 節と at-most-one 節を用いるのも直接符号化と同様である．

制約に関しては，違反点集合ではなく支持点集合に着目する．また，支持符号化は 2 変数間の制約のみを扱う [2]．例えば，制約 $x \leq y$ に関して，$x=3$ のときにこの制約を満たす y の値は 3 と 4 であるので，「$x=3$ ならば $y=3$ か $y=4$」を

[2] 多変数制約に拡張したものに [5] がある．

表す節 $\neg p(x=3) \vee p(y=3) \vee p(y=4)$ を追加する．また，$x=4$ のときに制約を満たす y の値は 4 であるため，節 $\neg p(x=4) \vee p(y=4)$ を追加する．

直接符号化と同様に，$x \geq 3$ と $y \leq 2$ は以下の節に符号化される．

$$\neg p(x=0) \quad \neg p(x=1) \quad \neg p(x=2)$$
$$\neg p(y=3) \quad \neg p(y=4)$$

単位伝播により $\neg p(x=3)$ と $\neg p(x=4)$ が導出されるが，これは at-least-one 節に違反しているため，即座に矛盾が検出される．

符号化後の節数および伝播能力について

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 $O(d)$ 個の変数と $O(d^2)$ 個の節が必要となる．また，各 2 項制約は $O(d^2)$ 個の節に符号化される．支持符号化は，3 項制約に対しては適用できない．したがって，小規模ドメインサイズの問題に対して約 10^7 個の節が必要となってしまい，中規模以上のドメインサイズの問題には実質的に適用できない．

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により，制約ソルバーでのアーク整合維持法 (MAC) を実現できる [27]．例に挙げたような 2 項制約間の矛盾は，決定変数を決める必要がなく単位伝播のみで検出できるため，効率的である．しかし，グラフ彩色問題による速度比較では，他の符号化よりも遅いという結果が得られている [66]．

4.2.3 対数符号化

対数符号化 (log encoding) は，整数の 2 進法表記に着目した SAT 符号化である [36, 25, 66]．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 $p(x^{(i)})$ を導入する．

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

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

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．

$$\begin{aligned} & (x^{(2)} + c_2 \leq y^{(2)}) \\ \wedge \quad & (x^{(2)} + c_2 \leq y^{(2)} - 1 \vee x^{(1)} + c_1 \leq y^{(1)} + 2c_2) \\ \wedge \quad & \cdots \end{aligned}$$

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に符号化するとCNF式にならないため，第2.2節で述べたTseitin変換を行い充足同値な以下の制約へ変換する (q, r は新しい命題変数)．

$$\begin{aligned} & (x^{(2)} + c_2 \leq y^{(2)}) \\ \wedge \quad & (q \vee r) \\ \wedge \quad & (\neg q \vee x^{(2)} + c_2 \leq y^{(2)} - 1) \\ \wedge \quad & (\neg r \vee x^{(1)} + c_1 \leq y^{(1)} + 2c_2) \\ \wedge \quad & \cdots \end{aligned}$$

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

$$\begin{array}{ll} \neg p(y^{(2)}) \vee p(c_2) & p(x^{(2)}) \vee \neg p(y^{(2)}) \\ p(x^{(2)}) \vee p(c_2) & q \vee r \\ \neg q \vee \neg p(y^{(2)}) & \neg q \vee p(c_2) \\ \neg q \vee p(x^{(2)}) & \cdots \end{array}$$

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

$$\begin{array}{c} p(x^{(2)}) \vee p(x^{(1)}) \\ \neg p(y^{(2)}) \\ \neg p(y^{(1)}) \vee \neg p(y^{(0)}) \end{array}$$

この例では単位伝播により，$\neg p(y^{(2)})$ 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．

また桁ごとの制約ではなく，支持符号化のように制約の支持点集合を考えて符号化する**対数支持符号化**も提案されている．

符号化後の節数および伝播能力について

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，乗算を含まない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・チェック法より弱い制約伝播となる [77]．SAT ソルバーでの単位伝播により MSB での範囲伝播が実現できるが，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，他の符号化よりも効率が悪くなる．

第5章　順序符号化

本章では，**順序符号化** (order encoding) を用いたSAT符号化と，順序符号化に基づくSAT型制約ソルバーSugarについて述べる．また，グラフ彩色問題，順序符号化を採用したSAT型制約ソルバーSugarの2008年度の制約ソルバー競技会の結果および釣合い型不完備ブロック計画 (Balanced Incomplete Block Design; ***BIBD***) 構成問題 を用いた性能評価の結果を示す．

5.1　整数変数の符号化

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 [63, 64, 66]．これは，Crawford と Baker がジョブショップ・スケジューリング問題に適用した方法 [15, 34, 47] をCSPに適用可能なように一般化したものである．

例えば，整数変数 $x \in \{0..4\}$ は，命題変数 $p(x \leq 0), p(x \leq 1), p(x \leq 2), p(x \leq 3)$ で表される．$x \leq 4$ は恒真であるため，$p(x \leq 4)$ 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．

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

例えば，$\neg p(x \leq 0) \vee p(x \leq 1)$ 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 5.1 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．

5.2　制約の符号化

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，$a_1 < x_1 \leq b_1$，$a_2 < x_2 \leq b_2$，…，$a_n < x_n \leq b_n$ 内のすべての点 $(x_1, x_2, \ldots, x_n)$ が制約を違反し

表 5.1: 充足する値割り当て

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

ている時，以下の節を追加する．

$$\begin{aligned} & p(x_1 \leq a_1) \vee \neg p(x_1 \leq b_1) \\ \vee \quad & p(x_2 \leq a_2) \vee \neg p(x_2 \leq b_2) \\ \vee \quad & \cdots \\ \vee \quad & p(x_n \leq a_n) \vee \neg p(x_n \leq b_n) \end{aligned}$$

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 [64]．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 $\sum_{i=1}^{n} a_i x_i \leq c$ は以下のように符号化される．

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

ここで b_i は $\sum_{i=1}^{n} b_i = c - n + 1$ を満たすように動くとし，変換 $()^{\#}$ は以下のように定義する．

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

ただし，x の最小値未満の場合には $p(x \leq a)$ を 0 に変換し，x の最大値以上については 1 に変換する．$x = y$ の形の制約は，$(x \leq y) \wedge (y \leq x)$ 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．

例えば，制約 $x + 1 \leq y$ は，以下の 5 つの節に符号化される．

$$\begin{gathered} \neg p(y \leq 0) \\ p(x \leq 0) \vee \neg p(y \leq 1) \\ p(x \leq 1) \vee \neg p(y \leq 2) \\ p(x \leq 2) \vee \neg p(y \leq 3) \\ p(x \leq 3) \end{gathered}$$

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

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

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

これらの節から単位伝播により $p(y \leq 2)$ 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 $\{x + 1 \leq y, x \geq 2, y \leq 2\}$ の矛盾が検出される．また，各 $p(x \leq i)$ および 各 $p(y \leq i)$ の順序関係を表す節から，$\neg p(x \leq 0)$，$p(y \leq 3)$ が真であることが導出できる．

一般にドメインサイズを 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．

5.3 符号化後の節数および伝播能力について

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適用可能である．

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，小規模及び中規模な問題では効率的な求解が可能である．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 [50]．

5.4 SAT 型制約ソルバー Sugar

SAT 型制約ソルバー Sugar は，整数有限領域上の線形の CSP を順序符号化に基づき SAT 符号化し，SAT ソルバーにより求解するシステムである [62, 65, 66]．SAT 符号化部分は Java で記述され，SAT ソルバーとしては MiniSat，PicoSAT 等が利用可能である．

また，符号化した SAT 問題を目的変数の範囲条件を変更しながら解くことにより，制約最適化問題および最大 CSP にも対応している．

Expression	Replacement	Extra condition
$E < F$	$E + 1 \leq F$	
$E = F$	$(E \leq F) \wedge (E \geq F)$	
$E \neq F$	$(E < F) \vee (E > F)$	
$\max(E, F)$	x	$(x \geq E) \wedge (x \geq F) \wedge ((x \leq E) \vee (x \leq F))$
$\min(E, F)$	x	$(x \leq E) \wedge (x \leq F) \wedge ((x \geq E) \vee (x \geq F))$
$\mathrm{abs}(E)$	x	$(x \geq E) \wedge (x \geq -E) \wedge ((x \leq E) \vee (x \leq -E))$
E div c	q	$(E = cq + r) \wedge (0 \leq r) \wedge (r < c)$
E mod c	r	$(E = cq + r) \wedge (0 \leq r) \wedge (r < c)$

図 5.1: $\sum a_i x_i \leq b$ 以外の式の変換

Sugar では，与えられた CSP を，前処理により一旦 CSP の CNF 式に変換している．CSP の CNF 式におけるリテラルは，$\sum a_i x_i \leq b$ の形の線形制約，外延的制約，命題変数，または命題変数の否定のいずれかである．

なお，線形制約 $\sum a_i x_i \leq b$ の形になっていない比較式や算術式は，図 5.1 に示す方法で変換する．図中，“Expression” が元の式，“Replacement” が置換後の式，“Extra condition” が追加する制約である．また，E div c および E mod c は，式 E を正整数定数 c で割った商と剰余を表す．

その他，以下の方法の導入により実用的な SAT 型制約ソルバーを実現している．

5.4.1 3 項制約への置換

第 5.2 節で述べたように，線形制約 $\sum_{i=1}^{n} a_i x_i \leq b$ は，一般には $O(d^{n-1})$ 個の節に符号化される．ここで，d はドメインサイズを表す．

しかし，新しい整数変数を導入すれば，線形制約中の変数の個数を 3 個以下にできるため，線形制約 $\sum_{i=1}^{n} a_i x_i \leq b$ は，$n \geq 4$ の場合でも $O(nd^2)$ 個の節に SAT 符号化できる．

例えば $a_1 x_1 + a_2 x_2 + a_3 x_3 + a_4 x_4 \leq b$ の場合，新しい整数変数 y を導入し，元の線形制約を $a_1 x_1 + a_2 x_2 + y \leq b$ に置き換えた上，2 個の新しい制約 $a_3 x_3 + a_4 x_4 - y \leq 0$，$-a_3 x_3 - a_4 x_4 + y \leq 0$ を追加すれば良い．この方法を繰り返すことによって，任意の線形制約について，変数の個数を 3 個以下にできる．

5.4.2 制約伝播によるドメインの縮小

ドメインのサイズを d とした時，d が非常に大きい場合は多数の節になる．そこで Sugar では，通常の制約ソルバーで行われているものと同様の制約伝播を行い，SAT 符号化の前にドメインの縮小を行っている．例えば，CSP ソルバー競技会のベンチマーク問題の 1 つである FISCHER11-6-fair については，制約伝播により，5.5 億以上の無駄なドメイン値が削除される．

5.4.3 グローバル制約の変換

グローバル制約については，基本的にはそれぞれの定義に従い同等の制約式に変換した後，SAT 符号化を行っている．

ただし，与えられた n この整数変数 $x_1, \ldots, x_n$ が互いに異なることを意味する alldifferent 制約については，まず，定義にしたがって $\bigwedge_{i<j}(x_i \neq x_j)$ なる制約に変換し，さらに $\neg\bigwedge(x_i < \ell(x_i)+n-1)$ および $\neg\bigwedge(x_i > u(x_i)-n+1)$ という制約を追加する ($\ell(x_i)$ および $u(x_i)$ はそれぞれ x_i の下限と上限)．これは，互いに異なる n 変数が $n-1$ のサイズのドメインに入らないことを表す鳩の巣原理 (pigeonhole principle) の条件を追加したことに相当し，大幅な性能向上を実現している．

alldifferent 制約についてはこれまで様々な整合性アルゴリズムが提案されている [76]．それらの内，範囲整合性 (bounds consistency) アルゴリズムは，n 個の整数変数が取りうるすべての範囲について鳩の巣原理による条件を利用する方法である．しかし，そのまま SAT 符号化した場合，膨大な節数となる．

Sugar で用いている上記の方法は，最大の範囲についてのみ鳩の巣原理を利用することで，追加する節数を 2 つだけにし，効率的な変換を実現している点が特徴となっている．

5.5 制約最適化問題への対応

COP (X, D, P, C, v) の最適値は，CSP を繰り返し解くことで得られる．

$$\min \{a \in D(v) \mid \text{CSP}(X, D, P, C \cup \{v \leq a\})\text{が充足可能}\}$$

COP の解は，a を [60, 34, 47] で提案されているように，二分探索を用いることで効率的に求められる．

P を COP とし，その目的変数を v とする．図 5.2 に，P の最適値を求める Sugar の最小化手続きを示す．`minimize(`P`,` v`)` の概要は以下のようになる．

1. P を SAT に符号化し，変数 S に代入する．S は SAT ファイルを表す．
2. 変数 found に false を代入する．found は，解が見つかったかどうかを表すフラグである．
3. 変数 lb と ub に v の下限と上限をそれぞれ代入する．
4. lb < ub が成り立つ時，ステップ (10) へ行く．
5. 変数 a に $\lfloor$(lb+ub)/2$\rfloor$ を代入する．
6. 変数 c に単位節 $p(v, \mathtt{a})$ を代入する．ここで，$p(v, \mathtt{a})$ は $v \leq \mathtt{a}$ を表す命題変数である．
7. MiniSat を S $\vee$ c を入力の SAT 問題として実行する．
8. ub を a に更新し，もし結果が充足可能であれば，found に true を代入する．そうでなければ，lb を a+1 に更新する．
9. ステップ (4) に戻る．
10. もし found が true であれば，この手続きは P の最適値を発見して成功し，そうでなければ失敗する．

Sugar は COP を最初に一度 SAT に符号化し，$v \leq a$ に対応する節のみを繰り返し変更する．しかし，以下のようにいくつかの改善すべき点がある．

- 最適値が得られるまで，何度も MiniSat のプロセスを起動する必要がある．
- MiniSat の各プロセスで生成された学習節を再利用できない．

学習節の再利用は，探索空間を大幅に減らすことができる．このため，学習節の再利用は非常に重要である．この問題を解決するため，Sugar を改良した Sugar++では，MiniSat のインクリメンタル機能を用いる．

Sugar++ が利用する MiniSat は，以下の 3 つの命令を標準入力から処理できるように変更されている．

- add $L_1\ L_2\ \cdots\ L_n$
  この命令は節 $\{L_1, L_2, \ldots, L_n\}$ を SAT 節データベースに追加する．この節は，最初に追加されたデータベースと同様に，探索の矛盾解析のために用いられる．
- solve $L_1\ L_2\ \cdots\ L_m$
  この命令は $L_1 \wedge L_2 \wedge \ldots \wedge L_m$ の仮定のもとで，SAT 問題の求解を行う．この仮定は MiniSat の solve メソッドの引数として渡され，探索中は一時的に真として扱われる．問題の求解後，この仮定は取り消される．この仮定は MisiSat の矛盾検出では用いられないため，生成される学習節には影響を与えない．

```
procedure minimize(P, v);
begin
  S := P の SAT符号化;
  found := false;
  lb := vの下限 ;
  ub := v + 1の下限+1;
  while lb < ub do
    a := ⌊(lb+ub)/2⌋;
    c := {p(v, a)};
    result := S ∨ c に対して MiniSat を実行した結果;
    if result が充足可能 then
      found := true;
      ub := a;
    else
      lb := a + 1;
    end if
  end while
  if found then
    OPTIMUM FOUND;
  else
    UNSATISFIABLE;
  end if
end
```

図 5.2: Sugar の最小化手続き

- `exit`
  この命令は MiniSat を終了させる.

以下が Sugar++の主な特徴である.

- Sugar++とインクリメンタル版の MiniSat の通信に，双方向 IO を用いる.
- MiniSat は一度だけ起動されるため，SAT ファイルの読み込みは一度だけである.
- 探索中の学習節は最利用される.

Sugar と比較して，Sugar++は SAT 問題の読み込みのオーバーヘッドを減らすだけでなく，探索中の学習節の再利用にも成功している.

5.5.1 最大 CSP への対応

最大 CSP は，COP に変換することで求解を行える.

```
procedure minimize(P, v);
begin
  S := P の SAT符号化;
  found := false;
  lb := v の下限;
  ub := v の上限 + 1;
  MiniSat のプロセスを S を引数として起動する;
  while lb < ub do
    a := ⌊(lb+ub)/2⌋;
    MiniSat に 'solve p(v, a)' 命令を送る;
    result := MiniSat の実行結果;
    if result が充足可能 then
      found := true;
      ub := a;
      MiniSat に 'add p(v, a)' 命令を送る;
    else
      lb := a + 1;
      MiniSat に 'add -p(v, a)' 命令を送る;
    end if
  end while
  MiniSat に 'exit' 命令を送る;
  if found then
    OPTIMUM FOUND;
  else
    UNSATISFIABLE;
  end if
end
```

図 5.3: Sugar++の新しい最小化手続き

Sugarでは，$C_1, C_2, \ldots, C_m$ を制約とする最大CSPが与えられた時，新しい整数変数 $p \in \{0, 1, \ldots, m\}$ および $p_i \in \{0, 1\}$ $(i = 1, 2 \ldots, m)$ を導入し，下記の制約の元で目的変数 p の値を最小化するCOPを構成する．

$$p \geq \sum_{i=1}^{m} p_i$$
$$(p_i > 0) \vee C_i \quad (i = 1, 2, \ldots, m)$$

ここで，p_i は制約 C_i が満たされない場合のペナルティを表している．

上記の最適解が元の最大CSPの最適解となる．

第6章　順序符号化の性能評価

本章では，グラフ彩色問題，順序符号化を採用したSAT型制約ソルバー Sugar の2008年度のCSP/最大CSPソルバー競技会の結果および釣合い型不完備ブロック計画 (Balanced Incomplete Block Design; ***BIBD***) 構成問題 を用いた性能評価の結果を示す．

6.1　グラフ彩色問題での比較

ここでは，グラフ彩色問題を使用した実験結果を通じて，各種SAT符号化の性能比較を行う．なおグラフ彩色問題のSAT符号化 については，文献 [52, 25, 64] などの研究がある．

グラフ彩色問題のベンチマーク問題としては，Graph Coloring and its Generalization[1]で公開されている全119問のうち，彩色数 (必要となる最小の色数 $\chi(G)$) が既知[2]の52問を用いた．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．

実験は，本解説で述べた直接, 多値, 支持, 対数, 対数支持, 順序の6種類の符号化を用い，色数 $k = \chi(G)$ の場合と $k = \chi(G) - 1$ の場合の2通りでSAT問題に符号化した624問について，それぞれを制限時間30分としてSATソルバーで解いた時の解けた問題数およびSATソルバーの使用したCPU時間を計測する方法で行った．

SATソルバーとしては，優れた性能で知られている MiniSat 2.0 [21] を使用し，Xeon 3GHz，メモリー 16GB の計算機上で実行した．

6.1にベンチマーク問題52問に対する集計結果を示す．平均CPU時間は，すべての符号化で解けた問題に対する値である ($k = \chi(G)$ の時46問，$k = \chi(G) - 1$ の時39問)．

色数 k が彩色数 $\chi(G)$ に一致する場合，6種類のいずれの符号化も30分以内に同一の46問について解を得た．平均CPU時間で見ると，多値と直接符号化が速く，順序符号化も比較的良い性能を示している．

[1] `http://mat.gsia.cmu.edu/COLOR04/`

[2] 各種符号化による予備実験で彩色数を決定したものを含む．

表 6.1: グラフ彩色問題の解けた問題数と平均 CPU 時間

	$k = \chi(G)$		$k = \chi(G) - 1$	
SAT 符号化	解けた問題数	平均 CPU 時間 (秒)	解けた問題数	平均 CPU 時間 (秒)
直接	46	0.07	42	13.61
多値	46	0.03	42	15.45
支持	46	1.77	39	63.74
対数	46	10.40	44	2.20
対数支持	46	4.40	44	2.04
順序	46	0.96	45	2.61

表 6.2: グラフ彩色問題 6 問の CPU 時間 (秒) の比較 ($k = \chi(G) - 1$)

グラフ彩色問題名	k	直接	多値	支持	対数	対数支持	順序
le450_15b	14	-	-	-	-	-	962.25
school1	13	-	-	-	1081.91	1081.91	535.15
school1_nsh	13	-	-	-	1541.99	1541.99	119.41
DSJR500.1	11	1770.00	858.71	-	7.13	7.13	2.05
queen8_12	11	1097.94	228.16	-	20.05	20.05	2.79
5-FullIns_4	8	642.10	670.70	-	305.51	305.51	27.89

表中の "-" は 1800 秒以内で解けなかったことを表す.

$k = \chi(G) - 1$, すなわち彩色不能な場合, 順序符号化が 45 問について充足不能を示し, 他のどの符号化よりも多くの問題を解いた. 45 問中, 他の符号化で解けない場合があった 6 問の CPU 時間を 6.2 に示す.

以上から, グラフ彩色問題について, 順序符号化が彩色可能な場合も彩色不能な場合も平均的に良い性能を示すといえる.

6.2 CSP/最大 CSP ソルバー競技会

CSP/最大 CSP ソルバー競技会は, 2008 年 6 月に開始され同年 9 月に結果が発表された [74].

提出された制約ソルバーは, 主催者の用意した実行環境で, 審判によって定められたそれぞれ 5 部門のベンチマーク問題に対して実行され, 性能が評価された.

実行環境は以下の通りである.

- CPU: Xeon 2GHz, 2MB cache, 32-bits mode
- CPU 制限時間: 30 分 (CSP), 60 分 (最大 CSP)
- メモリ制限: 900MB

表 6.3: CSP ソルバー競技会

部門名	問題数	ソルバー数
2-ARY-EXT	635	23
2-ARY-INT	696	22
N-ARY-EXT	704	24
N-ARY-INT	716	22
GLOBAL	556	17

表 6.4: 最大 CSP ソルバー競技会

部門名	問題数	ソルバー数
2-ARY-EXT	534	8
2-ARY-INT	276	6
N-ARY-EXT	278	8
N-ARY-INT	109	6
GLOBAL	98	2

性能は以下の方法で比較され，各部門における順位付けが行われた．

- 上記実行環境で解けた問題の個数による比較
  - CSP については，充足可能または充足不能を解答した問題の個数
  - 最大 CSP については，最適値を求めた問題の個数
- 解けた問題の個数が等しい場合は，CPU 時間による比較
- 一問でも間違った解答を行ったソルバーは，その部門で失格となり，評価の対象とならない．

表 6.3 および表 6.4 に，CSP/最大 CSP ソルバー競技会における部門名，問題数，その部門に参加したソルバー数を示す．CSP ソルバー競技会については合計で 14 チーム，24 ソルバーの参加，最大 CSP ソルバー競技会では 4 チーム，8 ソルバーの参加だった (各チームは 2 ソルバーまでを参加登録できる)．

各部門は，以下に示すようなベンチマーク問題から構成されている．

- **2-ARY-EXT 部門**: 2 変数間の外延的制約からなる問題．ほとんどが乱数生成されたランダム CSP の問題である．

表 6.5: CSP/最大 CSP ソルバー競技会の部門

	2 変数		多変数		
部門名	外延的	内延的	外延的	内延的	グローバル
2-ARY-EXT	○				
2-ARY-INT	○	○			
N-ARY-EXT	○		○		
N-ARY-INT	○	○	○	○	
GLOBAL	○	○	○	○	○

- **2-ARY-INT 部門**: 2 変数間の内延的制約および 2 変数間の外延的制約からなる問題．ショップ・スケジューリング，周波数割当，グラフ彩色，クイーン配置等の問題が含まれる．
- **N-ARY-EXT 部門**: 多変数間の外延的制約からなる問題．ランダム CSP，クロスワード等の問題が含まれる．
- **N-ARY-INT 部門**: 多変数間の内延的制約および外延的制約からなる問題．有界モデル検査，実時間相互排除プロトコル検証，マルチナップサック，擬似ブール制約，ゴロム定規，ソーシャルゴルファー等の問題が含まれる．
- **GLOBAL 部門**: グローバル制約および多変数間の内延的制約，外延的制約からなる問題．ラテン方陣，時間割作成等の問題が含まれる．

また，各部門の問題は以下のシリーズに分類されている．

- **REAL** (Real-World instances): 現実世界の応用例からなる問題．
- **PATT** (Patterned instances): 一定のパターンに従った問題 (ランダム生成を含む)．
- **ACAD** (Academic instances): ランダム生成を含まない学術的な問題．
- **QRND** (Quasi-random instances): 小規模の構造を含みランダムに生成された問題．
- **RAND** (Random instances): 純粋にランダムに生成された問題．
- **BOOL** (Boolean instances): ブール変数 (0-1 変数) のみを含む問題．

表 6.6: CSP ソルバー競技会での Sugar ソルバーの結果

	Category	Rank	#Solved	% of VBS
Sugar+minisat	2-ARY-EXT	14	470	76%
	2-ARY-INT	11	484	76%
	N-ARY-EXT	15	370	61%
	N-ARY-INT	12	486	74%
	GLOBAL	4	405	84%
Sugar+picosat	2-ARY-EXT	15	443	71%
	2-ARY-INT	10	486	77%
	N-ARY-EXT	16	347	57%
	N-ARY-INT	13	481	73%
	GLOBAL	1	424	85%

6.2.1 CSP ソルバー競技会の結果

CSP ソルバー競技会には，SAT ソルバーとして MiniSat を用いた Sugar+minisat と，PicoSat を用いた Sugar+picosat の 2 ソルバーで参加した [65].

表 6.6 に，CSP ソルバー競技会の各部門における Sugar ソルバーの結果を示す．“Rank” はその部門における順位，“#Solved” は解けた問題数，“% of VBS” は VBS(Virtual Best Solver) に対しての解いた問題数の割合を表す．VBS は，参加全ソルバーの最も良い結果を統合した仮想的なソルバーである．

Sugar ソルバーは，最も広い範囲の制約からなる GLOBAL 部門において，第 1 位および第 4 位という非常に優れた成績だった．

GLOBAL 部門以外の第 1 位は，すべて CPhydra というポートフォリオ型のソルバーであった．CPhydra は，内部に複数の制約ソルバーを持っており，CSP から抽出した特徴値からそれらの複数の制約ソルバーを実行する計画を作成し，その計画に基づいて内部の制約ソルバーを動作させ解を求めている [49]．今回の CPhydra は，競技会に同時参加した Mistral，choco，Abscon の 3 種類の制約ソルバーを用い，前回競技会のベンチマーク問題について，事前ベース推論を用いて事前に学習を行っている．

外延的制約が中心の問題の場合，制約には整数の順序関係が現れておらず，順序符号化による SAT 符号化が有効に働く場合が少ない．2-ARY-EXT および N-ARY-EXT の部門において，Sugar がやや低い順位となっているのは，このことが原因と考えられる．

2-ARY-INT 部門の結果について，第 1 位の CPhydra (597 問解答) と Sugar(486 問解答) を比較した所，グラフ彩色問題で 31 問，周波数割り当て問題で 72 問少な

表 6.7: GLOBAL 部門での解けた問題数の比較

Series		Sugar	CPhydra	Mistral	CaSPER	Choco
ACAD: BIBD	(83)	78*	70	67	57	51
ACAD: Costas Array	(11)	8	9	9	9	9
ACAD: Latin Square	(10)	9*	5	5	6	5
ACAD: Magic Square	(18)	8	8	8	16	6
ACAD: NengFa	(3)	3*	3	3	2	3
ACAD: Orthogonal Latin Square	(9)	3*	2	2	3	2
ACAD: Perfect Square Packing	(74)	53*	52	41	44	49
ACAD: Pigeons	(19)	19*	19	19	19	19
ACAD: Quasigroup Existence	(35)	29	28	28	30	28
BOOL: Pseudo-Boolean	(100)	70*	44	40	69	49
PATT: BQWH	(20)	20*	20	20	20	20
PATT: Cumulative Job-Shop	(10)	4*	2	2	2	1
PATT: RCPSP	(78)	78*	78	78	70	73
REAL: Cabinet	(40)	0	40	40	40	40
REAL: Timetabling	(46)	42*	40	41	10	3
TOTAL	(556)	424*	420	403	397	358

い解答数となっており，これらが差のほとんどを占めていた．グラフ彩色問題の制約はすべて等号否定 ($\neq$) であり，整数の順序関係が現れておらず，順序符号化による SAT 型ソルバーには不向きな問題といえる．周波数割当問題については，SAT 符号化した際に SAT 問題の規模が大きくなりすぎたためメモリオーバーとなっているものが多く，今後に課題を残していることがわかった．

N-ARY-INT の部門で，同様に第 1 位の CPhydra(569 問) と Sugar(486 問) を比較した所，クロスワード問題で 58 問，Primes 問題で 28 問少なく，差の大きな要因となっていた．クロスワード問題は外延的制約も多数含まれている問題であり，やはり Sugar に不向きな問題といえる．Primes 問題は比較的大きな素数を係数とする線形制約を制約としている問題であり，SAT 問題の規模が大きくなりメモリオーバーとなっているものが多く見られた．

最後に，GLOBAL 部門についての結果の詳細を表 6.7 に示し，図 6.1 に上位のソルバーの解いた問題数と CPU 時間のグラフを示す．GLOBAL 部門での順位は，第 1 位から第 9 位まで順に Sugar+picosat (424 問), CPhydra k_40 (420 問), CPhydra k_10 (419 問), Sugar+minisat (405 問), Mistral-prime (403 問), CaSPER zito (397 問), CaSPER zao (390 問), Mistral-option (383 問), Choco 2_dwdeg (358 問) であった．表 6.7 および図 6.1 では同一チームのソルバーについては上位のものだけを示して

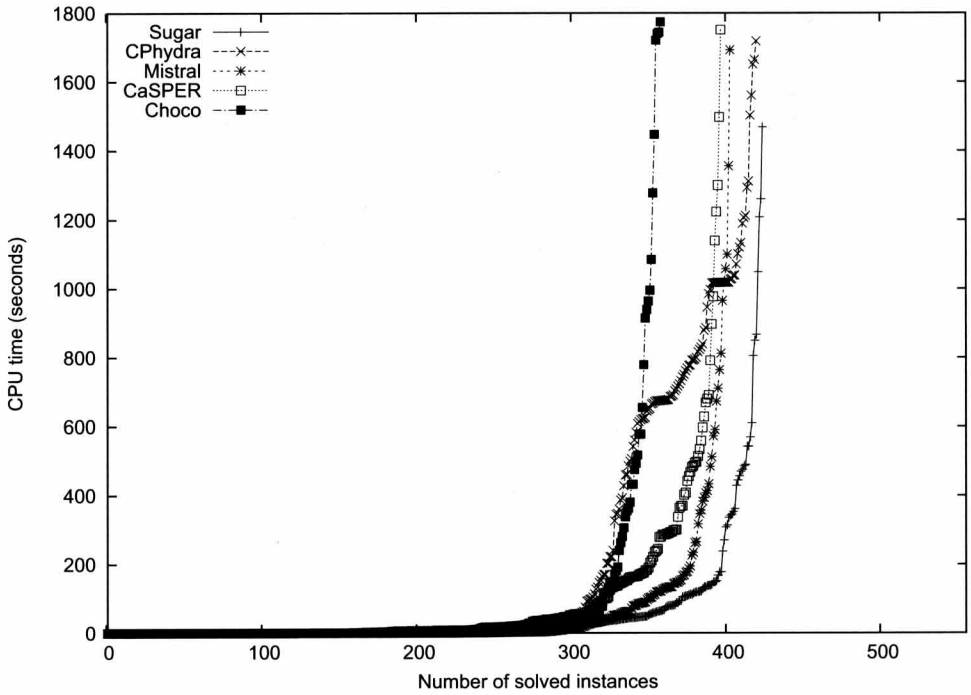

図 6.1: GLOBAL 部門: 制限時間内に解けた問題数

いる．また，表中 "Series" は問題のシリーズ名と総問題数である．また，Sugar 欄中の * 印は表中のソルバーのうちで最も良い結果であること表す．

Sugar は，GLOBAL 部門のほとんどのシリーズで最も優れた性能を示した．しかし，2 位の CPhydra および他のソルバーと比較すると，$\{0, 1610\}$ 等の 2 値のドメインを持つ変数の線形和が使用されている Cabinet 問題が全く解けていなかった．2 値のドメインを持つ変数自体は，順序符号化では 1 つのブール変数として効率良く符号化される．しかし，それらの線形和を符号化する際に新しい整数変数が導入され，そのドメインが大きくなるため，SAT 問題の規模も増大しメモリオーバーとなっていた．この点も今後の重要な検討課題である．

6.2.2 最大 CSP ソルバー競技会の結果

最大 CSP ソルバー競技会には，SAT ソルバーとして MiniSat を用いた Sugar と，第 5.5 節で述べた，MiniSat のインクリメンタル機能を利用する Sugar++の 2 ソルバーで参加した [65, 67]．

表 6.8 に，最大 CSP ソルバー競技会の各部門における Sugar および Sugar++ソルバーの結果を示す．

表 6.8: 最大 CSP ソルバー競技会での Sugar および Sugar++ソルバーの結果

	Category	Rank	#Solved	% of VBS
Sugar	2-ARY-EXT	2	240	55%
	2-ARY-INT	1	101	98%
	N-ARY-EXT	2	118	69%
	N-ARY-INT	1	39	93%
	GLOBAL	1	65	100%
Sugar++	2-ARY-EXT	3	229	52%
	2-ARY-INT	2	99	96%
	N-ARY-EXT	3	118	69%
	N-ARY-INT	2	39	93%
	GLOBAL	2	50	77%

参加したソルバーは多くはないが，Sugar および Sugar++ソルバーは，2-ARY-INT，N-ARY-INT，GLOBAL 部門で第 1 位と第 2 位，2-ARY-EXT，N-ARY-EXT 部門で第 2 位と第 3 位という非常に優れた成績であった．ただし，GLOBAL 部門には Sugar と Sugar++以外のソルバーは参加していない．

2-ARY-EXT および N-ARY-EXT 部門の第 1 位は，それぞれ toulbar2 および toulbar2/BTD という外延的制約のみを対象としたソルバーである [57]．なお，toulbar2 は N-ARY-EXT 部門で，toulbar2/BTD は 2-ARY-EXT 部門で失格になったため，それらの部門での順位は与えられていない．

6.2.3 考察

CSP ソルバー競技会で，参加 24 ソルバー中 9 ソルバーがいずれかの部門で失格となっていることからもわかるように，高性能かつ信頼性の高い制約ソルバーを開発することは簡単な仕事ではない．

Sugar は，競技会における数千問のベンチマーク問題に対し間違った解答を行うことがなく，その点で性能および信頼性の高い制約ソルバーといえる．

以下では，これまでの記述と重複する点もあるが，CSP/最大 CSP ソルバー競技会の結果について考察事項をまとめる．

- Sugar で用いている順序符号化は，広く用いられている直接符号化と比較して，よりコンパクトな SAT 符号化を実現している．しかし，大規模な問題については，依然として符号化後の SAT 問題が巨大となり，競技会の実行環境でメモリオーバーとなっていた．64 ビット CPU やより大きなメモリの

表 6.9: Sugar+minisat と Sugar+picosat の比較

	Sugar+minisat			Sugar+picosat		
Category	SAT+UNSAT	SAT	UNSAT	SAT+UNSAT	SAT	UNSAT
2-ARY-EXT	470	278	192	443	280	163
2-ARY-INT	484	257	227	486	261	225
N-ARY-EXT	370	179	191	347	178	169
N-ARY-INT	486	399	87	481	393	88
GLOBAL	405	252	153	424	273	151
TOTAL	2215	1365	850	2181	1385	796

利用が一般的になれば，自然に解消される問題ともいえるが，符号化方法の工夫により解決できる可能性もあり，今後の重要な研究課題の一つである．

- グローバル制約について，alldifferent 制約への鳩の巣原理の導入以外には特別な処理を行っていないにもかかわらず，CSP ソルバー競技会の GLOBAL 部門で第 1 位であった．詳細は今後分析する必要があるが，順序符号化の有効性が明確になったといえる．

- CSP ソルバー競技会での Sugar+minisat と Sugar+picosat の成績を比較すると，2-ARY-INT と GLOBAL 部門で Sugar+picosat のほうが上位，その他の部門では Sugar+minisat のほうが上位となった．表 6.9 に示すように，問題の充足可能性の別に分類して比較すると，充足可能な問題では Sugar+picosat が優れ，充足不能な問題では Sugar+minisat が優れていた．これは，頻繁なリスタートにより充足可能な SAT 問題での性能向上を実現した PicoSAT の効果によるものといえる．

- 最大 CSP ソルバー競技会では Sugar が Sugar++よりも優れた結果だった．Sugar++は，MiniSat のインクリメンタル探索機能を利用することにより，効率的な求解を目指している．しかし，そのため MiniSat のプロセスを終了させることなく継続して動作させており，MiniSat のメモリ使用量が増大する．それが原因となり競技会の実行環境では，メモリオーバーが生じていた．メモリ消費量を抑えながらインクリメンタル探索を実現する方法の検討が必要である．

6.3 釣合い型不完備ブロック計画構成問題

釣合い型不完備ブロック計画 (Balanced Incomplete Block Design; *BIBD*) [13] はラテン方格と並ぶ，組合せデザイン分野の代表的な問題である．$BIBD$ の応用分野としては，実験計画，符号理論，暗号理論などがある．

本節では，$BIBD$ を構成する問題に対して，新しいSAT符号化を提案する．この符号化は，二分木を用いて順序符号化を改良したものである．順序符号化と比較して，基数制約 (cardinality constraint) の符号化に必要な節数が少なくてよい点が特長である．

6.3.1 釣合い型不完備ブロック計画

v，b，r，k，λ を正の整数とし，v 個の元から成る集合 $\mathbb{P}$ と，$\mathbb{P}$ のいくつかの k 点部分集合からなる集合 $\mathbb{B}$ の組 $(\mathbb{P}, \mathbb{B})$ を**結合構造**と呼ぶ．$BIBD(v,b,r,k,\lambda)$ は，以下のように定義される．

定義 4. $BIBD(v,b,r,k,\lambda)$ は，以下の条件を満たす結合構造である．

- $\mathbb{P}$ の異なる2点を含むブロックの数が λ である．
- $\mathbb{P}$ の任意の点を含むブロックの数が r である．
- ブロックの総数は b である．
- $2 < k < v$ で，$\mathbb{B}$ が $\mathbb{P}$ の k 点部分集合全体の真部分集合となる．

例 1. $BIBD(4,4,3,3,2)$ の例を示す．

$$\begin{aligned} \mathbb{P} &= \{1,2,3,4\} \\ \mathbb{B} &= \{\{1,2,3\},\{1,2,4\},\{1,3,4\},\{2,3,4\}\} \end{aligned}$$

$BIBD(v,b,r,k,\lambda)$ の点とブロックを $\mathbb{P} = \{p_1,\ldots,p_v\}$，$\mathbb{B} = \{B_1,\ldots,B_b\}$ とし，$p_i \in B_j$ のとき，$a_{i,j} = 1$，そうでないとき，$a_{i,j} = 0$ と定義する．この時，$v \times b$ 行列 $A = (a_{i,j})$ を $BIBD(v,b,r,k,\lambda)$ の**結合行列**と呼ぶ．6.2に例1の $BIBD(4,4,3,3,2)$ の結合行列を示す．これは $v = 4$，$b = 4$ の 4×4 行列であり，各行の和が $r = 3$，各列の和が $k = 3$，相異なる2つの行の内積が $\lambda = 2$ であることがわかる．

***BIBD* 構成問題**とは，与えられた v，b，r，k，λ に対し，$BIBD(v,b,r,k,\lambda)$ が存在するかどうかを判定し，存在する場合は $BIBD(v,b,r,k,\lambda)$ を構成する問題である．$BIBD$ 構成問題は求解困難な組合せ問題であり，数多くの未解決問題が残されている [13]．最近では $BIBD(22,33,12,8,4)$ が存在しないことが，計算機による探索で示された．

	$\{1,2,3\}$	$\{1,2,4\}$	$\{1,3,4\}$	$\{2,3,4\}$
1	1	1	1	0
2	1	1	0	1
3	1	0	1	1
4	0	1	1	1

図 6.2: $BIBD(4,4,3,3,2)$ の結合行列

6.3.2 *BIBD* 構成問題の CSP 表現

$BIBD$ 構成問題の CSP 表現について述べる．この CSP 表現は $BIBD(v,b,r,k,\lambda)$ の結合行列に基づいており，1 つの行列と 3 種類の制約から構成される．

基本行列 は二値変数を要素とする $v \times b$ 行列である．各要素 $x_{i,j}$ $(1 \leq i \leq v,\ 1 \leq j \leq b)$ は，結合行列の各要素を表し，そのドメインは $x_{i,j} \in \{0,1\}$ である．

行制約 は“基本行列の各行の和が r” を表す制約である．

$$\sum_{j=1}^{b} x_{i,j} = r \quad (1 \leq i \leq v) \tag{6.1}$$

列制約 は“基本行列の各列の和が k” を表す制約である．

$$\sum_{i=1}^{v} x_{i,j} = k \quad (1 \leq j \leq b) \tag{6.2}$$

内積制約 は“基本行列の相異なる 2 つの行の内積が λ” を表す制約である．

$$\sum_{j=1}^{b} x_{i,j} \cdot x_{i',j} = \lambda \quad (1 \leq i < i' \leq v) \tag{6.3}$$

内積制約 6.3 は乗算 $x_{i,j} \cdot x_{i',j}$ を含むが，新しい二値変数 $y_{i,i',j} \in \{0,1\}$ $(1 \leq i < i' \leq v,\ 1 \leq j \leq b)$ を導入することにより，以下の制約に置き換えることができる．

$$y_{i,i',j} = 1 \Leftrightarrow (x_{i,j} = 1 \wedge x_{i',j} = 1) \tag{6.4}$$

$$\sum_{j=1}^{b} y_{i,i',j} = \lambda \quad (1 \leq i < i' \leq v) \tag{6.5}$$

本節では $BIBD$ 構成問題の CSP 表現として，制約 6.1 6.2 6.4 6.5 を用いる．この CSP 表現の主要な制約 6.1 6.2 6.5 は，**基数制約** $\sum_{i=1}^{n} X_i = c$ $(X_i \in \{0,1\}, c$ は

整数定数) で表される．したがって，SAT 符号化を用いて $BIBD$ 構成問題を効率よく解くためには，基数制約の SAT 符号化が重要な位置を占める．

本節で述べた CSP 表現は，前節で述べた国際 CSP ソルバー競技会のベンチマークにも使用されてる一般的なものであり，特定のソルバーに対して有利な表現ではない．他の制約モデルとしては，Meseguer と Torras のモデル [42]，Prestwich のモデル [51] などがある．

6.3.3 $BIBD$ の順序符号化

$BIBD$ 構成問題の CSP 表現を順序符号化を用いて SAT 問題に符号化する方法を述べる．二値変数 $x_{i,j}$，$y_{i,i',j}$ に対し，命題変数 $p(x_{i,j} \leq 0)$，$p(y_{i,i',j} \leq 0)$ を導入する (ただし，$1 \leq i < i' \leq v, \quad 1 \leq j \leq b$)．

制約 6.4 は以下の節に符号化される．

$$p(y_{i,i',j} \leq 0) \quad \vee \quad \neg p(x_{i,j} \leq 0) \tag{6.6}$$

$$p(y_{i,i',j} \leq 0) \quad \vee \quad \neg p(x_{i',j} \leq 0) \tag{6.7}$$

$$\neg p(y_{i,i',j} \leq 0) \quad \vee \quad p(x_{i,j} \leq 0) \vee p(x_{i',j} \leq 0) \tag{6.8}$$

6.6，6.7 は $y_{i,i',j} = 1 \Rightarrow (x_{i,j} = 1 \wedge x_{i',j} = 1)$ を，6.8 は $y_{i,i',j} = 1 \Leftarrow (x_{i,j} = 1 \wedge x_{i',j} = 1)$ を符号化したものである．

行制約 6.1 は，まず線形比較の連言 $\left(\sum_{j=1}^{b} x_{i,j} \leq r\right) \wedge \left(\sum_{j=1}^{b} x_{i,j} \geq r\right)$ に置き換えられる．その後，各線形比較は前節で述べた $\sum_{i=1}^{n} a_i z_i \leq c$ と同じ方法で符号化される．列制約 6.2，内積制約 6.5 も同様に符号化される．

この符号化の欠点は，基数制約 6.1，6.2，6.5 の符号化に必要な節数が巨大になることである．例えば，行制約 6.1 の符号化には $O(v2^{b-1})$ 個の節が必要となる．この問題を回避するために，基数制約に対する新しい順序符号化を提案する．

6.3.4 二分木を用いた基数制約の順序符号化

基数制約 $\sum_{i=1}^{n} X_i = c$ ($X_i \in \{0,1\}$，c は整数定数) を二分木を用いて分解した後，順序符号化を用いて符号化する手法を提案する．

基数制約 $\sum_{i=1}^{n} X_i = c$ に対して，以下のように二分木を生成する．ラベル値 n の孤立点から始めて，ラベル値 $m \geq 2$ の各終端点に対して，ラベル値が $\lfloor m/2 \rfloor$ と $m - \lfloor m/2 \rfloor$ の 2 つの子節点を接続する操作を再帰的に繰り返す．この操作により，ラベル値 1 の葉を n 個もつ二分木が生成される．続いて，この二分木の葉に二値変数 $X_i (1 \leq i \leq n)$ を割り当てる (全単射)．根にはドメイン $\{0..n\}$ の新しい整数

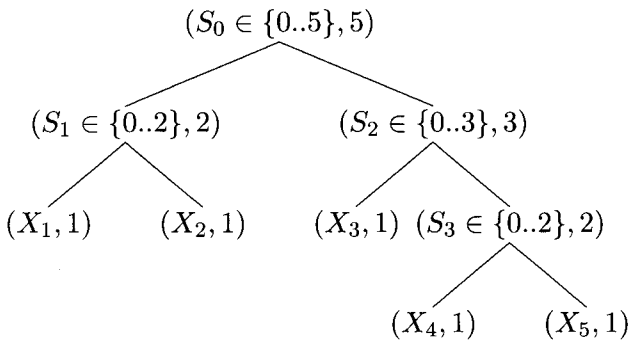

図 6.3: $\sum_{i=1}^{5} X_i = 3$ $(X_i \in \{0, 1\})$ に対する二分木と制約の例

表 6.10: 基数制約 $\sum_{i=1}^{n} X_i = c$ $(X_i \in \{0, 1\}$, c は整数定数) の符号化に必要な節数と補助変数の数

手法	節数	補助変数の数
順序符号化	$O(2^{n-1})$	0
提案手法	$O(n^2 + n \log n)$	$O(n \log n)$

変数を割り当てる．ラベル値 m の各内点にドメイン $\{0..m\}$ の新しい整数変数を割り当てる．次に，二分木を基に新しい制約を生成する．根に割り当てられた変数 S_0 に対して，制約 $S_0 = c$ を生成する．親節点の変数 S_p とその 2 つの子節点の変数 S_q，S_r に対して，制約 $S_p = S_q + S_r$ を生成する．最後に，生成された変数および制約を順序符号化を用いて符号化する．

6.3 に，基数制約 $\sum_{i=1}^{5} X_i = 3$ $(X_i \in \{0, 1\})$ に対して生成される二分木と制約の例を示す．各節点の左側が割り当てられた整数変数，右側がラベル値を表している．根に割り当てられた変数 S_0 は基数制約の左辺 $\sum_{i=1}^{5} X_i$ に対応し，各内点に割り当てられた変数 $S_j (1 \leq j \leq 3)$ はその部分和を表している．

表 6.10 に基数制約 $\sum_{i=1}^{n} X_i = c$ の SAT 符号化に必要な節数と補助変数の数を示す．順序符号化は $O(2^{n-1})$ 個の節が必要であるのに対し，提案手法では $O(n^2 + n \log n)$ 個と少ない節数に抑えることができる．補助変数とは，提案手法において二分木の内点に割り当てた整数変数を符号化するために必要な命題変数である．順序符号化には必要ないが，提案手法ではこれらの整数変数の符号化に $O(n \log n)$ 個の補助変数が必要となる．

提案手法の有効性 (特に，求解の効率性と拡張性) について述べる．符号化の効

率性については，解く問題の性質および利用するSATソルバーの特性(系統的ソルバーか確率的ソルバーか)によって異なるため明確な基準はないが，SAT符号化の研究者の間では以下の二つの基準がよく用いられる．

1. 符号化後のSAT問題に対するSATソルバーの単位伝播と元のCSPに対する制約伝播との関連

2. 符号化に必要な節数

(1)については，提案手法のベースとなる順序符号化は第5.3節で述べたように，SATソルバーの単位伝播が元のCSPに対する範囲伝播に対応しており，直接符号化，支持符号化，対数符号化等の他の符号化と比較して，効率の良い求解が可能である．(2)については，表6.10に示したように，提案手法は従来の順序符号化と比較して少ない節数に抑えることができる．さらに，提案手法は基数制約だけでなく擬似ブール制約 $\sum_{i=1}^{n} a_i X_i = c$ (a_i は非零の整数定数, $X_i \in \{0,1\}$, c は整数定数)，線形制約 $\sum_{i=1}^{n} a_i z_i = c$ (a_i は非零の整数定数，z_i は相異なる整数変数，c は整数定数) にも適用可能であり拡張性が高い．

基数制約は古くから研究されているが，ここ数年新しいSAT符号化がいくつか提案されている．Bailleuxらの符号化 [1] は，単位伝播がアーク整合性維持法に対応し，基数制約 $\sum_{i=1}^{n} X_i \leq c$ の符号化に $O(n^2)$ の節数を必要とする．Sinzの符号化 [59] とCodishらの符号化 [11] は，必要な節数が各々$O(nc)$，$O(n \log^2 c)$ に抑えられる．Bailleuxらの符号化は，二分木を用いる点で提案手法と類似しているが，順序符号化とは異なる．また，これらの符号化は基数制約専用であるため，擬似ブール制約および線形制約に直接適用することはできない．Enらによる擬似ブール制約の符号化 [22] は，高速な擬似ブール制約ソルバー MiniSat+に実装され広く用いられている．提案手法と既存の符号化との比較実験および考察に関しては今後の課題とする．なお，6.3.5の $BIBD$ 構成問題(全237問)を用いて，提案手法とBailleuxらの符号化を比較した結果，解けた問題数は提案手法が1問多かった．

6.3.5 実行実験

提案手法の有効性を評価するため，$BIBD(v,b,r,k,\lambda)$ 構成問題 ($v \times b \leq 1400$, $3 \leq k \leq v/2$, 全237問) を用いて比較実験を行った．CSP表現には6.3.2で述べた制約6.1 6.2 6.4 6.5を使用した．比較に用いたシステムは，以下の通りである．

- 提案手法を用いてSAT符号化し，高速SATソルバー MiniSat 2.2 core [3]を用いて求解．

[3] http://minisat.se/Main.html

表 6.11: 解けた問題数の比較

$v \times b$	問題数	提案手法	順序	Mistral	choco
49-100	3	**3**	**3**	**3**	**3**
101-200	11	**11**	**11**	**11**	**11**
201-300	12	**12**	**12**	**12**	**12**
301-400	17	**16**	**16**	**16**	15
401-500	14	**13**	12	**13**	7
501-600	19	**17**	**17**	**17**	7
601-700	19	**17**	**17**	16	6
701-800	20	**16**	15	10	4
801-900	19	**16**	15	10	5
901-1000	16	**11**	10	9	4
1001-1100	27	**22**	**22**	6	3
1101-1200	20	**16**	15	4	4
1201-1300	25	**17**	16	4	3
1301-1380	15	**14**	13	2	4
合計	237	**201**	194	133	88

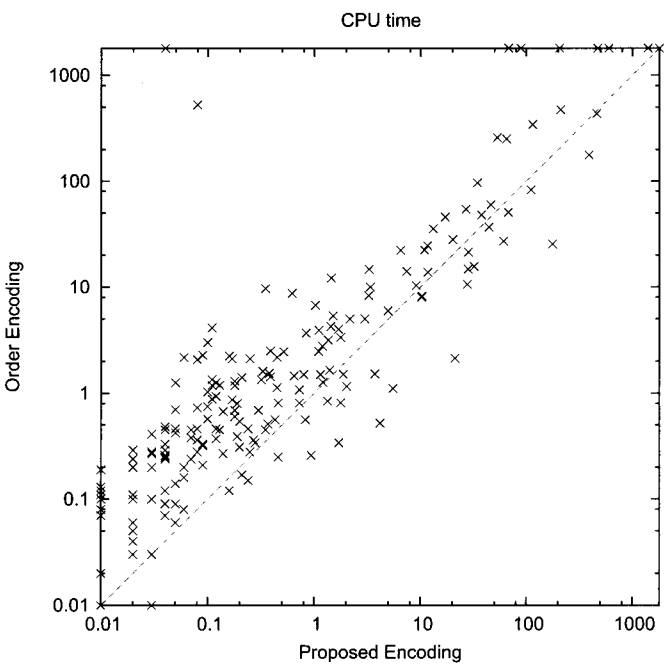

図 6.4: CPU 時間の比較

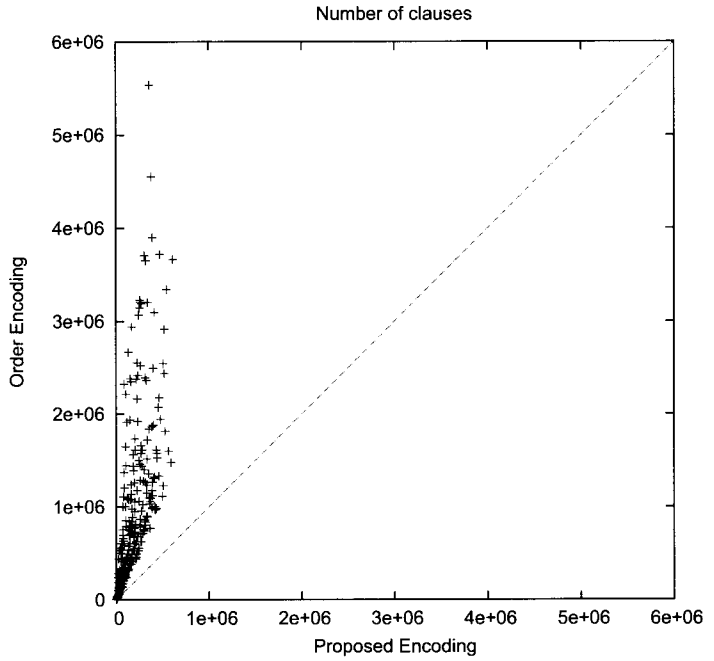

図 6.5: 節数の比較

- 順序符号化を用いて SAT 符号化し，MiniSat 2.2 core を用いて求解．節数を抑えるため，Sugar のヒューリスティックスを用いて基数制約を分解．
- 高速制約ソルバー Mistral 1.550[32] を用いて求解．
- 高速制約ソルバー choco (choco2_impwdeg)[72] を用いて求解．

各システムにおいて，SAT ソルバー／制約ソルバーの解が充足可能 (SAT) の場合は，その解から $BIBD$ を構成することができる．また，解が充足不能 (UNSAT) の場合は，$BIBD$ を構成できないことを意味する．Mistral と choco については，制約 6.4 を $y_{i,i',j} = x_{i,j} \cdot x_{i',j}$ で置き換え，各基数制約の記述にグローバル制約の一つである weightedSum を用いた．実験環境は Linux マシン (Intel Xeon 3.00GHz, メモリ 8GB) であり，各ソルバーのタイムアウトは 1800 秒とした．

まず最初に，表 6.11 に各システムで解けた問題数を示す．左から順に，$v \times b$ の値の範囲，問題数，解けた問題数を表している．各 $v \times b$ について解けた問題数が最も多いシステムの値をボールド体で示す．提案手法は全 237 問中 201 問と最も多くの問題を解いている．提案手法で解けた問題数は順序符号化より 7 問多く，順序符号化で解けた問題は全て提案手法でも解けている．また，$v \times b$ の値が大きくなると，Mistral と choco は解ける問題数が大きく減るのに対して，提案手法は安

定的に問題を解いている．なお，各システムとも解けた問題はすべてSAT，すなわち，$BIBD$を構成できる場合であった．

次に，提案手法と順序符号化について，MiniSatが求解に要したCPU時間の比較結果を6.4に示す．横軸が提案手法，縦軸が順序符号化のCPU時間を表している(横軸，縦軸ともに対数目盛)．6.4より，多くの問題に対して提案手法が順序符号化より高速に求解していることがわかる．より詳細には，提案手法が順序符号化よりも高速に求解した問題数は165問であり，その逆は30問であった．また，解けた問題のCPU時間の幾何平均は，提案手法が0.28秒，順序符号化が0.76秒であり，提案手法が約2.7倍高速に求解している．

最後に，提案手法と順序符号化について，生成されるSAT問題の節数の比較結果を6.5に示す．横軸が提案手法，縦軸が順序符号化によって生成されたSAT問題の節数を表している．6.5から，提案手法の節数は，順序符号化と比較して，圧倒的に少ないことがわかる．例えば，比較的規模の大きな$BIBD(27, 27, 13, 13, 6)$の符号化後の節数は，順序符号化が2418228に対して，提案手法は232443と約10分の1に抑えられる．

なお，他の制約モデル(およびSAT符号化)として，Prestwichのモデル[51]の性能も測定したが，解ける問題数が非常に少ないため，実験結果は省略した．

6.3.6 対称解の除去

$BIBD$構成問題は対称性が高く，対称解が多く存在する．例えば，$BIBD(4, 4, 3, 3, 2)$の結合行列(6.2)を見ると，任意の相異なる2つの行(あるいは列)を入れ替えたものも解となることがわかる．制約プログラミングの分野において，対称解を除去することは**対称性除去 (symmetry breaking)**と呼ばれ，重要な研究課題となっている．一般に対称性除去は解探索のコストを減らす効果があるが，SAT符号化に対する有効性については十分に研究されているとはいえない．

本章では，提案手法に対する対称性除去の有効性を評価するために，以下の手法を用いて実験を行った．これらの手法は結合行列の行と列に関する対称解を除去するものであり，存在すべき解を失うものではない．実験環境，ベンチマーク，タイムアウトは，6.3.5と同じである．

- Double Lex [23]：基本行列(6.3.2)の連続する相異なる2つの行，および連続する相異なる2つの列に対して辞書式順序制約を適用する．
- Frischの手法 [24]：Double Lexに加え，基本行列の1，2行目を固定する．3行目以降の各行に対して，固定した1，2行目との内積制約から導かれる制約を追加する．

表 6.12: 提案手法に対称性除去を適用した結果：解けた問題数

問題数	なし	Double	Frisch	Snake
237	201	202	**203**	200

表 6.13: 提案手法に対称性除去を適用し UNSAT の問題を解いた結果：CPU 時間 (単位：秒)

v	b	r	k	λ	なし	Double	Frisch	Snake
15	21	7	5	2	T.O.	8314	6042	**2541**
21	28	8	6	2	T.O.	T.O.	T.O.	T.O.
22	22	7	7	2	T.O.	T.O.	62104	**4917**

- Snake Lex [30]：Double Lex の改良として提案されたスネーク順序に基づく手法である．

表 6.12 に提案手法に対称性除去手法を適用して解けた問題数を示す．問題数に続いて，対称性除去なし，Double Lex，Frisch の手法，Snake Lex を適用して解けた問題数を表している．Double Lex の拡張である Frisch の手法が 203 問と最も多くの問題を解いた (対称性除去なしと比べて 3 問増・1 問減)．新たに解けた 3 問は，$v \times b$ の値が各々578，1083，1224 と比較的大きく，MiniSat が要した CPU 時間は 1208.67 秒，1002.29 秒，512.93 秒であった．これは，追加制約が解の探索空間を削減し，解の発見に有効に作用したと考えられる．(どの対称性除去手法を用いても) 解けなくなった 1 問は，MiniSat がバックトラックなしで 0.04 秒と短時間で解いていることから，非常に特殊なケースといえる．追加制約により MiniSat の変数選択順序および学習節に変化が生じ，解けなくなったものと考えられる．

6.3.5 の実験結果と同様，解けた問題はすべて SAT であり，UNSAT の問題は 1 問も解くことができなかった．そこでベンチマーク中から $BIBD$ が構成できない (UNSAT となる) ことが理論的に証明されている問題を 3 問選び，タイムアウトを 1800 秒から 24 時間に延ばして同様の実験を行った．表 6.13 に CPU 時間を示す．“T.O.” は 24 時間以内に求解できなかったことを表す．Frisch の手法と Snake Lex が 3 問中 2 問の UNSAT を示すことができた．

第7章　コンパクト順序符号化

本章では，前述した各符号化の問題点を指摘し，それらの問題点を解決する符号化として提案するコンパクト順序符号化について述べる．

7.1　既存のSAT符号化の問題点

図 7.1 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．

既存の符号化の問題点は以下のようにまとめられる．

符号化後のサイズ:　直接符号化と支持符号化は中規模な問題に対して約 $10^7 \sim 10^{15}$ 個以上の節が必要となる．したがって中規模以上の問題には適用できない．また，順序符号化は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，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．

制約ソルバーの伝播アルゴリズムとの関係:　対数符号化での単位伝播は，フォワード・チェック法よりも弱い制約伝播となり [77]，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．

[1]表では，筆算を符号化する方法での計算量を示している．高速フーリエ変換を用いることで，節数を $O(m \log m)$ まで減らすことができる．

表 7.1: 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)[1]	$O(\log^2 d)$	$O(\log^2 d)$
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

7.2 基本アイデア

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 [69, 70, 68, 71].

- 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 ($B \geq 2$, $m = \lceil \log_B d \rceil$, すべての x_i に対し $0 \leq x_i < B$). 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 $O(mB)$ 個の変数が導入される．また後述のように，各 2 項制約は $O(mB)$ 個の節に，各加算制約，乗算制約は $O(mB^2)$ 個と $O(mB^3 + m^2B^2)$ 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場合には，大規模な問題に対しても，$m = 3$ を選択すれ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，$m = 2$ 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，$m = 5$ 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．

7.3 コンパクト順序符号化の概要

以下では，まず 3 項 CSP (3ary-CSP)，制限 3 項 CSP (Restricted 3ary-CSP; R-CSP)，コンパクト 3 項 CSP (Compact 3ary-CSP; C-CSP) を定義する．R-CSP は 3 項 CSP から C-CSP への還元を簡単化するためのものであり，取りうる算術制約の形を制限した 3 項 CSP である．C-CSP は各整数変数の上限を $B - 1$ に制限した CSP である．

コンパクト順序符号化は，CSP を C-CSP へ還元し，還元された C-CSP を順序符号化することで実現できる．CSP の順序符号化については既に第 5 節で述べているため，以下では CSP から C-CSP への還元までを示す．

まず，各算術制約に含まれる整数変数の数を高々3 個に制限した (整数有限領域上の)**3 項 CSP** を以下のように定義する．

定義 5 (3 項 CSP)**.** (整数有限領域上の) 3 項 CSP は，以下を満たす組 (X, u, P, C) である．

- X は整数の有限集合である．
- u は X から $\mathbb{Z}$ への写像であり，整数変数の上限を表す (下限は 0 に固定されている)．
- P は命題変数の有限集合である．
- C は，X と P 上の内延的制約であり，以下の文法で表される．

$$C ::= p \mid \neg p \mid \sum_{i=1}^{n} a_i x_i \triangleright b \mid z = xy \mid C \wedge C \mid C \vee C$$

ただし $p \in P$, $n \in \{1,2,3\}$, $x_i, x, y, z \in X$, $a_i \in \mathbb{Z}$, $b \in \mathbb{N}$, $\triangleright \in \{\leq, \geq, =\}$ とする.

任意の整数有限領域上の CSP は，以下の方法で 3 項 CSP へ還元できる.

- 外延的制約を内延的制約を用いて表す. 例えば，整数変数 $x, y \in \{1,2,3\}$ 上の支持点集合 $R = \{(1,2),(2,3)\}$ からなる外延的制約は，内延的制約 $(x = 1 \wedge y = 2) \vee (x = 2 \wedge y = 3)$ で表すことができる.
- 第 5.4.3 節と同様の方法で，グローバル制約を内延的制約を用いて表す.
- 整数変数 $x \in X$ と，$\ell(x) \leq v \leq u(x)$ かつ $v \notin D(x)$ となる整数 v に対して，制約 $x \neq v$ を C に追加する ($\ell(x)$ と $u(x)$ はそれぞれ x の下限と上限を表す).
- 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.
- 新たな整数変数 z_i を導入することで，制約 $\sum_i a_i \prod_j x_{ij} \triangleright b$ を線形和 $\sum_i a_i z_i \triangleright b$ と積 $z_i = \prod_j x_{ij}$ へ還元する.
- 部分積を表す新しい変数を導入することで，3 変数以上の積を 2 変数の積に置き換える.
- 部分和を表す新しい変数を導入することで，4 変数以上の線形和を 3 変数の線形和に置き換える.

以下では (整数領域上の有限) 3 項 CSP を単に CSP と呼ぶ.

7.4 制限 3 項 CSP

説明の簡単化のため，3 項 CSP の取りうる制約を制限した**制限 3 項 CSP** (Restricted 3ary-CSP; **R-CSP**) を以下のように定義する.

定義 6 (制限 3 項 CSP (R-CSP)). R-CSP は制約 C を以下の形に制限された 3 項 CSP(X, u, P, C) である.

$$\begin{aligned} C \ ::= \ & p \mid \neg p \mid x \leq a \mid x \geq a \mid x \leq y \mid z = x + a \mid z = x + y \\ & \mid z = ax \mid z = xy \mid C \wedge C \mid C \vee C \end{aligned}$$

ただし $p \in P$, $x, y, z \in X$, $a \in \mathbb{N}$ である.

R-CSP の取りうる制約の形は非常に限定されているが，以下の補題が示すように，任意の CSP の制約は，R-CSP の制約へ還元できる．

補題 1. 任意の CSP (X, u, P, C) は R-CSP へ還元できる．

これは，制約 $\sum_{i=1}^{n} a_i x_i \triangleright b$ のすべての場合の還元を検証することで示すことができる．また，R-CSP の解から新たに導入された変数への値割り当てを単に取り除くことで，元の CSP の解を復元できる．

7.5 コンパクト3項CSP

この節では，**コンパクト 3 項 CSP** (Compact 3ary-CSP; **C-CSP**) を定義する．コンパクト 3 項 CSP は，各整数変数の上限を整数 $B-1$ に固定した CSP である．また，コンパクト 3 項 CSP に含まれる算術制約はすべて線形制約である．

定義 7 (コンパクト 3 項 CSP (C-CSP))．C-CSP は，以下を満たす組 $(B; X, P, C)$ である．

- $B \geq 2$ は基底を表す整数である．
- X は整数変数の有限集合である．ただし，任意の $x \in X$ に対して $u(x) = B-1$ である (下限は 0 に固定されている)．
- P は命題変数の有限集合である．
- 論理式 C は以下の文法で表される．

$$C \ ::= \ p \mid \neg p \mid \sum_{i=1}^{n} \pm x_i \triangleright b \mid By + z = ax \mid C \wedge C \mid C \vee C$$

ただし，$p \in P$，$n \in \{1, 2, 3\}$，$x_i, x, y, z \in X$，$0 \leq a < B$，$0 \leq b \leq 3(B-1)$，$\triangleright \in \{\leq, \geq, =\}$ である．

例 2. u と C が以下のように与えられる C-CSP$(\{x^{(1)}, x^{(0)}, y^{(1)}, y^{(0)}\}, u, \emptyset, C)$ を考える．

- $u(x^{(i)}) = u(y^{(i)}) = 9 \quad (i \in \{0, 1\})$
- $C = (x^{(1)} \leq y^{(1)}) \wedge (x^{(1)} \leq y^{(1)} - 1 \vee x^{(0)} \leq y^{(0)})$

この C-CSP の充足可能性は 図 7.1 のように表される．これは $10x^{(1)} + x^{(0)} \leq 10y^{(1)} + y^{(0)}$ と充足同値である．

x_1	x_0	Satisfiable
0–1	0–9	Yes
2	0–6	Yes
2	7–9	No
3–9	0–9	No

図 7.1: $(x_1 \leq 2) \wedge (x_1 \leq 1 \vee x_0 \leq 6)$ の充足可能性

7.6 R-CSP から C-CSP への還元

第 7.4 節で述べたように，任意の CSP は R-CSP へと還元できる．よって以下では，任意の基底 $B \geq 2$ に対して，R-CSP から C-CSP へ還元することを考える．R-CSP の還元を示す前に，いくつかの記法を導入する．まず，**最大ドメインサイズ** d および**桁数** m は以下のように定義される．

$$\begin{aligned} d &= 1 + \max(\{u(x) \mid x \in X\} \cup \{a \mid a \text{ は } C \text{ 中に現れた整数定数}\}) \\ m &= \lceil \log_B d \rceil \end{aligned}$$

また $x^{(i)}$ を，R-CSP の整数変数記号と整数 i $(0 \leq i < m)$ に対して，x の各桁を表す C-CSP の新しい整数変数記号を生成する構文上の関数として定義する．また，整数定数 $a \in \mathbb{N}$ に対しても同様に $a^{(i)}$ という記法を用いる．また，以下では記法 $x^{((j,i))}$ を $\sum_{k=i}^{j} B^{k-i} x^{(k)}$ として定義する．ただし，x は変数もしくは定数である．

R-CSP の制約のうち，$x \leq y$，$x \leq a$，$x \geq a$ は同様の方法で還元できる．$z = x+y$ と $z = x + a$，$z = xy$ と $z = ax$ も同様の方法で還元できるため，以下では $x \leq y$，$z = x + y$，$z = xy$ の還元のみを示す．

7.6.1 $x \leq y$ の還元

R-CSP の任意の論理式 $x \leq y$ は，C-CSP の整数変数を用いて $x^{(m-1,0)} \leq y^{(m-1,0)}$ と表現される．整数変数の上限を変えずに，この式を同値な C-CSP の制約に還元する方法を以下に示す．

定義 8 ($x \leq y$ の還元)．x と y を R-CSP の整数変数あるいは定数とすると，構文上の関数 $\tau(x, y)$ は以下のように定義される．

$$\begin{aligned} \tau(x,y) &:= \tau_{m-1}(x,y) \\ \tau_0(x,y) &:= x^{(0)} \leq y^{(0)} \\ \tau_i(x,y) &:= x^{(i)} \leq y^{(i)} \wedge (x^{(i)} \leq y^{(i)} - 1 \vee \tau_{i-1}(x,y)) \qquad (i > 0) \end{aligned}$$

命題 1. C-CSP の任意の整数変数あるいは整数定数 $x^{(i)}$, $y^{(i)}$ について以下が成り立つ.

$$x^{(m-1,0)} \leq y^{(m-1,0)} \iff \tau(x,y)$$

これは m に関する帰納法により証明できる.

例 3. 以下に $\tau_2(x,y)$ の例を示す. これは $x^{(2,0)} \leq y^{(2,0)}$ (すなわち $B^2x^{(2)} + Bx^{(1)} + x^{(0)} \leq B^2y^{(2)} + By^{(1)} + y^{(0)}$) と同値である.

$$\begin{aligned}
& \tau_2(x,y) \\
= \; & x^{(2)} \leq y^{(2)} \wedge (x^{(2)} \leq y^{(2)} - 1 \vee \tau_1(x,y)) \\
= \; & x^{(2)} \leq y^{(2)} \wedge (x^{(2)} \leq y^{(2)} - 1 \vee (x^{(1)} \leq y^{(1)} \wedge (x^{(1)} \leq y^{(1)} - 1 \vee \tau_0(x,y)))) \\
= \; & x^{(2)} \leq y^{(2)} \wedge (x^{(2)} \leq y^{(2)} - 1 \vee (x^{(1)} \leq y^{(1)} \wedge (x^{(1)} \leq y^{(1)} - 1 \vee x^{(0)} \leq y^{(0)})))
\end{aligned}$$

7.6.2 $z = x + y$ の還元

R-CSP の任意の論理式 $z = x + y$ は, C-CSP の整数変数を用いて $z^{(m-1,0)} = x^{(m-1,0)} + y^{(m-1,0)}$ と表される. 整数変数の上限を変えずに, この式を充足同値な C-CSP の制約に還元する方法を以下に示す.

定義 9 ($z = x + y$ の還元). $B \geq 2$ を基底, z, x, y を R-CSP の整数変数か定数とし, c_i $(0 \leq i \leq m)$ を他の場所に現れない命題変数とする. この時, 構文上の関数 $\sigma(z,x,y)$ は以下のように定義される.

$$\begin{aligned}
\sigma(z,x,y) \quad := \quad & \neg c_0 \wedge \neg c_m \\
\wedge \quad & \bigwedge_{i=0}^{m-1} ((c_i \vee c_{i+1} \vee z^{(i)} = x^{(i)} + y^{(i)}) \\
& \wedge (c_i \vee \neg c_{i+1} \vee z^{(i)} = x^{(i)} + y^{(i)} - B) \\
& \wedge (\neg c_i \vee c_{i+1} \vee z^{(i)} = x^{(i)} + y^{(i)} + 1) \\
& \wedge (\neg c_i \vee \neg c_{i+1} \vee z^{(i)} = x^{(i)} + y^{(i)} + 1 - B))
\end{aligned}$$

$\sigma(z,x,y)$ は C-CSP の 3 項の算術制約しか含んでおらず, その制約数は m に比例する.

命題 2. $B \geq 2$ を基底, $z^{(i)}$, $x^{(i)}$, $y^{(i)}$ を C-CSP の整数変数あるいは整数定数とする. この時, 任意の α について以下が成り立つ.

$$(\alpha, \emptyset) \models z^{(m-1,0)} = x^{(m-1,0)} + y^{(m-1,0)} \iff \exists \beta.(\alpha, \beta) \models \sigma(z,x,y)$$

Proof. 桁ごとの加算を考えることで，$z^{(m-1,0)} = x^{(m-1,0)} + y^{(m-1,0)}$ は $z^{(i)}$，$x^{(i)}$，$y^{(i)}$ に対する任意の値割り当てと $c'_i \in \{0,1\}$ $(0 \leq i \leq m)$ に対するある値割り当てに対して $c'_0 = 0 \wedge c'_m = 0 \wedge \bigwedge_{i=0}^{m-1} Bc'_{i+1} + z^{(i)} = x^{(i)} + y^{(i)} + c'_i$ と充足同値であることが容易に確認できる．$c'_i = 0$ と 1 のすべての場合を考え，各 c'_i を命題変数 c_i で置き換えることで，$z^{(i)}$，$x^{(i)}$，$y^{(i)}$ に対する任意の値割り当てに対して $z^{(m-1,0)} = x^{(m-1,0)} + y^{(m-1,0)}$ と $\sigma(z,x,y)$ が充足同値であることを示すことができる． □

例 4. 以下に基底 $B = 2$，$u(z) = u(x) = u(y) = 7$ のときの $\sigma(z,x,y)$ の例を示す．これは3ビット整数の加算を表している．

$$
\begin{aligned}
&\sigma(z,x,y) \;=\; \\
&\quad \neg c_0 \wedge \neg c_3 \\
&\wedge\ (c_0 \vee c_1 \vee z^{(0)} = x^{(0)} + y^{(0)}) \wedge (c_0 \vee \neg c_1 \vee z^{(0)} = x^{(0)} + y^{(0)} - 2) \\
&\wedge\ (\neg c_0 \vee c_1 \vee z^{(0)} = x^{(0)} + y^{(0)} + 1) \wedge (\neg c_0 \vee \neg c_1 \vee z^{(0)} = x^{(0)} + y^{(0)} - 1) \\
&\wedge\ (c_1 \vee c_2 \vee z^{(1)} = x^{(1)} + y^{(1)}) \wedge (c_1 \vee \neg c_2 \vee z^{(1)} = x^{(1)} + y^{(1)} - 2) \\
&\wedge\ (\neg c_1 \vee c_2 \vee z^{(1)} = x^{(1)} + y^{(1)} + 1) \wedge (\neg c_1 \vee \neg c_2 \vee z^{(1)} = x^{(1)} + y^{(1)} - 1) \\
&\wedge\ (c_2 \vee c_3 \vee z^{(2)} = x^{(2)} + y^{(2)}) \wedge (c_2 \vee \neg c_3 \vee z^{(2)} = x^{(2)} + y^{(2)} - 2) \\
&\wedge\ (\neg c_2 \vee c_3 \vee z^{(2)} = x^{(2)} + y^{(2)} + 1) \wedge (\neg c_2 \vee \neg c_3 \vee z^{(2)} = x^{(2)} + y^{(2)} - 1)
\end{aligned}
$$

7.6.3 $z = xy$ の還元

R-CSP の任意の論理式 $z = xy$ は，C-CSP の整数変数を用いて $z^{(m-1,0)} = x^{(m-1,0)} y^{(m-1,0)}$ と表される．一般の場合の還元を考える前に，特殊な場合である $z = ay$ $(0 \leq a < B)$ を考える．

$z = ay$ $(0 \leq a < B)$ の還元

$z = ay$ を y の各桁について分解し，各 $ay^{(i)}$ を v_i と表現する．この時，$z = ay$ は以下のように表される．

$$\bigwedge_{i=0}^{m-1} (v_i = ay^{(i)}) \quad \wedge \quad z = \sum_{i=0}^{m-1} B^i v_i$$

$ay^{(i)} < B^2$ なので，各 v_i は $Bv_i{}^{(1)} + v_i{}^{(0)}$ と表される．

定義 10 ($z = ay$ $(0 \leq a < B)$ の還元). $B > 2$ を基底，z と y を R-CSP の整数変数または整数定数，$0 \leq a < B$ を整数定数，v_i $(0 \leq i < m)$ を他の場所に現れない整数変数とする．この時，構文上の関数 $\nu(z, a, y)$ は以下のように定義される．

$$\nu(z, a, y) \quad := \quad \bigwedge_{i=0}^{m-1} (Bv_i{}^{(1)} + v_i{}^{(0)} = ay^{(i)}) \wedge \left[z = \sum_{i=0}^{m-1} B^i v_i \right]$$

ただし $[e]$ は左シフト演算と命題 2 を繰り返して用いて e を還元する構文上の関数とする．

補題 2. $B > 2$ を基底，$z^{(i)}$ と $y^{(i)}$ を C-CSP の整数変数または整数定数，$0 \leq a < B$ を整数定数とする．この時任意の α に対して以下が成り立つ．

$$(\alpha, \emptyset) \models z^{(m-1,0)} = ay^{(m-1,0)} \iff \exists \alpha'. \exists \beta. (\alpha \cup \alpha', \beta) \models \nu(z, a, y)$$

ただし α' は ν によって新しく導入された変数のみを含む値割り当てである．

Proof. 容易に確認できるため省略する． □

<u>一般の $z = xy$ の還元</u>

$z = xy$ を x の各桁について分解し，各 $x^{(i)}$ を w_i と置くことで，$z = xy$ は以下のように表される．

$$\bigwedge_{i=0}^{m-1} (w_i = x^{(i)} y) \quad \wedge \quad z = \sum_{i=0}^{m-1} B^i w_i$$

各 $x^{(i)}$ は 0 から $B-1$ の値しか取らず，$x^{(i)} = a$ の時 $w_i = ay$ なので，各 $w_i = x^{(i)} y$ は以下で表現できる．

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

ay の計算は，様々な i に対し繰り返して現れる．冗長な計算を除去するため，各 a に対して新しい変数 $y_a = ay$ を導入することで，$z = xy$ は以下のように表される．

$$\bigwedge_{i=0}^{m-1} \bigwedge_{a=0}^{B-1} \left((x^{(i)} \leq a - 1) \vee (x^{(i)} \geq a + 1) \vee (w_i = y_a) \right)$$
$$\wedge \quad z = \sum_{i=0}^{m-1} B^i w_i \wedge \bigwedge_{a=0}^{B-1} y_a = ay$$

ここで，各 $w_i = y_a$ は $\bigwedge_{j=0}^{m-1} (w_i{}^{(j)} = y_a{}^{(j)})$ へ還元でき，各 $y_a = ay$ は $\nu(y_a, a, y)$ へ還元できる．

定義 11 ($z = xy$ の還元). $B > 2$ を基底，z，x，y は R-CSP の整数変数もしくは整数定数，y_a $(0 \leq a < B)$ と w_i $(0 \leq i < m)$ は他の場所に現れない整数変数とする．この時，構文上の関数 $\mu(z, x, y)$ は以下のように定義される．

$$\begin{aligned}\mu(z,x,y) \quad := \quad & \bigwedge_{i=0}^{m-1} \bigwedge_{a=0}^{B-1} \left((x^{(i)} \leq a-1) \vee (x^{(i)} \geq a+1) \vee \bigwedge_{j=0}^{m-1} ({w_i}^{(j)} = {y_a}^{(j)}) \right) \\ \wedge \quad & \left[z = \sum_{i=0}^{m-1} B^i w_i \right] \wedge \bigwedge_{a=0}^{B-1} \nu(y_a, a, y)\end{aligned}$$

ただし $[e]$ は左シフト演算と命題 2 を繰り返し用いることで e を還元する構文上の関数とする．

命題 3. $B > 2$ を基底，$z^{(i)}$，$x^{(i)}$，$y^{(i)}$ を C-CSP の整数変数もしくは整数定数とする．この時，任意の α に対して以下が成り立つ．

$$(\alpha, \emptyset) \models z^{(m-1,0)} = x^{(m-1,0)} y^{(m-1,0)} \iff \exists \alpha'. \exists \beta. (\alpha \cup \alpha', \beta) \models \mu(z, x, y)$$

ただし α' は μ によって新しく導入された変数のみを含む値割り当てである．

Proof. 容易に確認できるため省略する． □

例 5. 以下に基底 $B = 2$，$u(z) = u(x) = u(y) = 3$ のときの $\mu(z, x, y)$ の例を示す．これは 2 ビット整数の乗算を表す．

$$\begin{aligned}\mu(z,x,y) \quad = \quad & ((x^{(1)} \geq 1) \vee ({w_1}^{(0)} = {y_0}^{(0)} \wedge {w_1}^{(1)} = {y_0}^{(1)})) \\ \wedge \quad & ((x^{(1)} \leq 0) \vee ({w_1}^{(0)} = {y_1}^{(0)} \wedge {w_1}^{(1)} = {y_1}^{(1)})) \\ \wedge \quad & ((x^{(0)} \geq 1) \vee ({w_0}^{(0)} = {y_0}^{(0)} \wedge {w_0}^{(1)} = {y_0}^{(1)})) \\ \wedge \quad & ((x^{(0)} \leq 0) \vee ({w_0}^{(0)} = {y_1}^{(0)} \wedge {w_0}^{(1)} = {y_1}^{(1)})) \\ \wedge \quad & [z = Bw_1 + w_0] \wedge \nu(y_1, 1, y) \wedge \nu(y_0, 0, y)\end{aligned}$$

7.6.4 全体の還元

これまでの結果を用いることで，R-CSP 全体の還元を示すことができる．

定義 12. 任意の基底 $B \geq 2$ と R-CSP の論理式 C に対して，関数 C^* を以下のように定義する．

$$\begin{array}{rclrclrcl}(p)^* & = & p & (x \leq y)^* & = & \tau(x,y) & (z = xy)^* & = & \mu(z,x,y) \\ (\neg p)^* & = & \neg p & (z = x + a)^* & = & \sigma(z,x,a) & (C_1 \wedge C_2)^* & = & C_1^* \wedge C_2^* \\ (x \leq a)^* & = & \tau(x,a) & (z = x + y)^* & = & \sigma(z,x,y) & (C_1 \vee C_2)^* & = & C_1^* \vee C_2^* \\ (x \geq a)^* & = & \tau(a,x) & (z = ax)^* & = & \mu(z,a,x) & & & \end{array}$$

命題 4. (X, u, P, C) を R-CSP とする．$B \geq 2$ を基底，$(B; X', P', C')$ を以下で定義される C-CSP とする．

$$\begin{aligned} X' &= \{x^{(i)} \mid x \in X,\ 0 \leq i < m\} \cup \{x \mid x \text{ は } C' \text{ に新しい導入された整数変数}\} \\ P' &= P \cup \{p \mid p \text{ は } C' \text{ に新しい導入された命題変数}\} \\ C' &= C^* \wedge \bigwedge_{x \in X} (x \leq u(x))^* \end{aligned}$$

この時以下が成り立つ．すなわち R-CSP と C-CSP は充足同値である．

$$\exists(\alpha, \beta) \models (X, u, P, C) \iff \exists(\alpha', \beta') \models (B; X', P', C')$$

Proof. 命題 1，2，3 から示される． □

最終的に，任意の CSP は C-CSP へ還元できる．

定理 2 (CSP から C-CSP への還元)**.** 任意の CSP は，任意の基底 $B \geq 2$ に対して充足同値な C-CSP へ還元できる．

Proof. 補題 1 と命題 4 から示される． □

7.7 CSP のコンパクト順序符号化

元の CSP を還元した C-CSP に順序符号化を適用することで，CSP のコンパクト順序符号化を実現できる．第 5 節で述べたように，順序符号化ではドメインサイズ d の各整数変数は $O(d)$ 個の命題変数で表され，n 項制約 $\sum_{i=1}^{n} a_i x_i \triangleright b$ は $O(d^{n-1})$ 個の節に符号化される $(\triangleright \in \{\leq, \geq, =\})$．

前節で述べたように，任意の CSP は任意の基底 B の C-CSP に還元できる．また C-CSP の各整数変数の上限は $B-1$ に制限されており，各算術制約は高々3 つの整数変数しか含まない．

$\tau(x, y)$ が $O(\log_B d)$ 個の比較を含み，各比較が $O(B)$ 個の節に符号化されるため，各比較制約 $x \leq y$ は $O(B \log_B d)$ 個の節に符号化される．同様に，各加算制約 $z = x + y$ は $O(B^2 \log_B d)$ 個の節に符号化される．$\nu(z, a, y)$ は $O(\log_B d)$ 個の乗算および加算を含み，それぞれの制約は $O(B^2)$ 個の節に符号化されるため，各乗算制約 $z = ay$ $(0 \leq a < B)$ は一般に $O(B^2 \log_B d)$ 個の節に符号化される．特別な場合として，$B \geq d$ の時は $O(d)$ 個の節に符号化される．$\mu(z, x, y)$ は $O(B)$ 回の ν の適用と $O(\log_B d)$ 個の加算を含む．各 ν は $O(B^2 \log_B d)$ 個の節に符号化され，各加算は $O(B^2 \log_B d)$ 個の節に符号化されるため．各 $z = xy$ は一般に $O(B^3 \log_B d + B^2 \log_B^2 d)$ 個の節に符号化される．特別に場合として，$B \geq d$ の時は ν が $O(d)$ 個の節に符号化されるため，$z = xy$ は $O(d^2)$ 個の節に符号化される．

表 7.2: 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)	$O(mB)$	$O(mB^2)$
提案 (乗算)	$O(m^2B^2)$	$O(mB^3 + m^2B^2)$

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
提案	MSD での範囲伝播

7.8 符号化後の節数および伝播能力について

図 7.1 にコンパクト順序符号化を追加したものが図 7.2 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，m は桁数を表す．

コンパクト順序符号化では，加算制約 $z = x + y$ は $O(mB^2)$ 個の節に，乗算制約 $z = xy$ は $O(mB^3 + m^2B^2)$ 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，より効率的な求解が可能となると考えられる．

7.9 関連研究

Sugar 以外の SAT 型制約ソルバーとしては，FznTini [33]，Bumblebee [43] と SAT4J CSP [40] がある．FznTini は対数符号化を，Bumblebee は順序符号化を，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 Eén と Sörensson は，擬似ブール制約を SAT に符号化する方法を提案している [22]．擬似ブール制約は，各変数のドメインが $\{0, 1\}$ に制限されている CSP である．Eén らの方法では，各整数変数を位取り基数法で表現し，各桁の表現に順序符号化に似た 1 進法を用いる．また Eén らは，より一般的な基底を可変にした基数法も提案している．

命題変数 $p(x \leq a)$ を使用する研究には [3, 4, 26, 31] がある．しかし，算術制約から $p(x \leq a)$ を用いた節への符号化は，最初に [15] で提案され，これはその後任意の線形制約に適用可能なように拡張された [63, 64]．

第8章　コンパクト順序符号化の性能評価

コンパクト順序符号化の有効性を検証するため，以下のベンチマークを用いて性能評価を行った．

- **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 ($m \in \{2,3\}$) および順序符号化，既存のSAT符号化である対数符号化，国際CSP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速度を比較した．

- **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速度を比較した．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，求解速度を比較した．

8.1　中規模な問題を用いた比較

中規模かつ実際的な問題として，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を用いて，コンパクト順序符号化 ($m \in \{2,3\}$) および順序符号化，既存のSAT符号化である対数符号化，制約ソルバーであるchoco，Mistralの求解数および求解速度に関して比較する．

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を決定することである．

ベンチマーク問題として，Brucker *et al.* [10] 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として，2009 年度国際 CSP ソルバー競技会上位ソルバーである choco 2.11 [72] と Mistral 1.550 [32] とも比較を行った．

表 8.1 に制限時間 1 時間 (3600 秒)，MiniSat 2.0 core [21] ソルバーが求解にかかった時間を示す (単位は秒)．実行環境は小規模の時と同様である．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に対して，コンパクト順序符号化は他の符号化や既存の制約ソルバーと比べても優れているといえる．

8.2 大規模な問題を用いた比較

大規模かつ実際的な問題として，先ほど用いた 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 ($m \in \{2, 3\}$) と先ほど用いた各ソルバーを求解数および求解速度に関して比較した．またコンパクト順序符号化 ($m \in \{2, 3\}$) と順序符号化，対数符号化に関して，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 対数符号化に関して，10^{10} 程度のドメインサイズを持つ超大規模な OSS を用いて，求解速度を比較した．

表 8.1: 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**

ベンチマークには，各 OSS 問題の処理時間を c 倍して生成した 102 問を用いた (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 8.2 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 8.1 に，各ソルバーでの求解速度の比較を示す．横軸は求解問題数，縦軸に求解速度を表している．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．図 8.2 に，いずれかの符号化で解けた問題に関して，各符号化での平均求解時間を示す．解けなかった問題は，解くのに 1 時間 (3600 秒) かかったものとした．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．

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速度は コンパクト順序符号化 ($m = 3$) と同等であるが，それより大きな d では求解数および求解速度が悪化し，$d \geq 10^6$ 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 ($m = 3$) は対数符号化より求解数が多い．求解速度の面に関しても，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 2.5 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 8.1 から，求解速度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 8.1 から，求解速度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 8.3 に，各符号化での矛盾が起きるまでに選択した決定変数の数の平均を示す．$c \leq 10^2$ 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

表 8.2: 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

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，$c = 10^4$ のときでもすべて 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．

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 3 章で示した目安だと，$m = 5$ 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，求解速度を比較した．結果，コンパクト順序符号化 ($m = 5$) は 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 ($m = 5$) の約 10 倍の時間が掛かった．

以上から，大規模かつ実際的な問題に対して，コンパクト順序符号化が既存の SAT 符号化や他の制約ソルバーよりも優れているといえる．また，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．

8.3 考察

上記の実験結果から，コンパクト順序符号化は小規模な問題から大規模な問題の全てに適用可能であるといえる．また 1 問についてではあるが，ドメインサイ

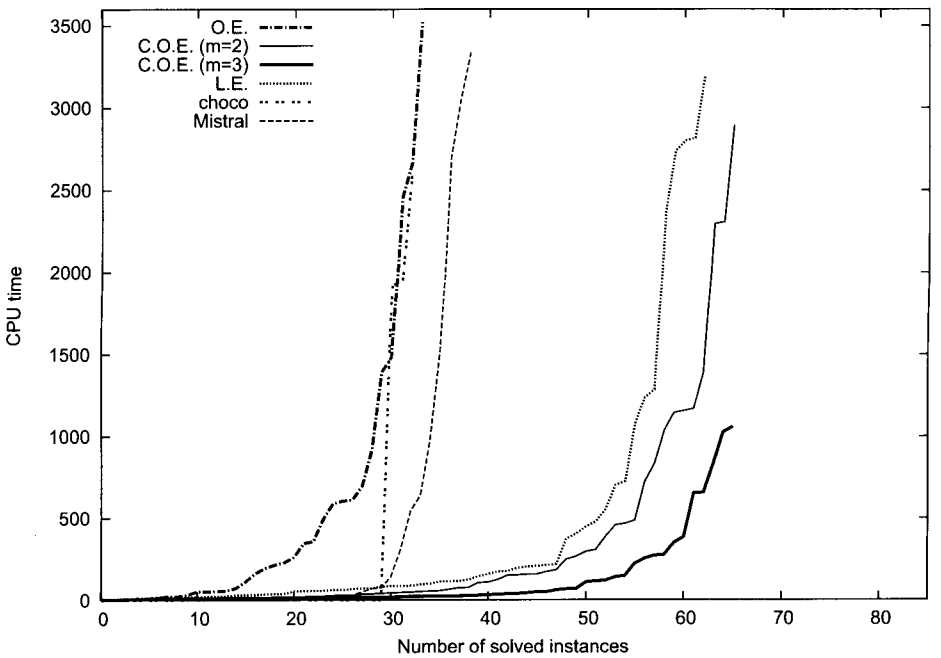

図 8.1: OSS ベンチマークにおける求解速度の比較

ズが 10^{10} 程度の超大規模な問題を含めて，コンパクト順序符号化は対数符号化よりも常に優れていることが確認できた．

また，中規模な問題に関しては，コンパクト順序符号化 $(m = 2)$ が優れた性能を示した．大規模な問題では，コンパクト順序符号化 $(m = 3)$ が最も優れた性能を示した．

本論文では示さなかったが，小規模な問題に関してもコンパクト順序符号化は順序符号化と比較してそれほど性能が落ちておらず，直接符号化や支持符号化より性能が良いことを確認した．

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．

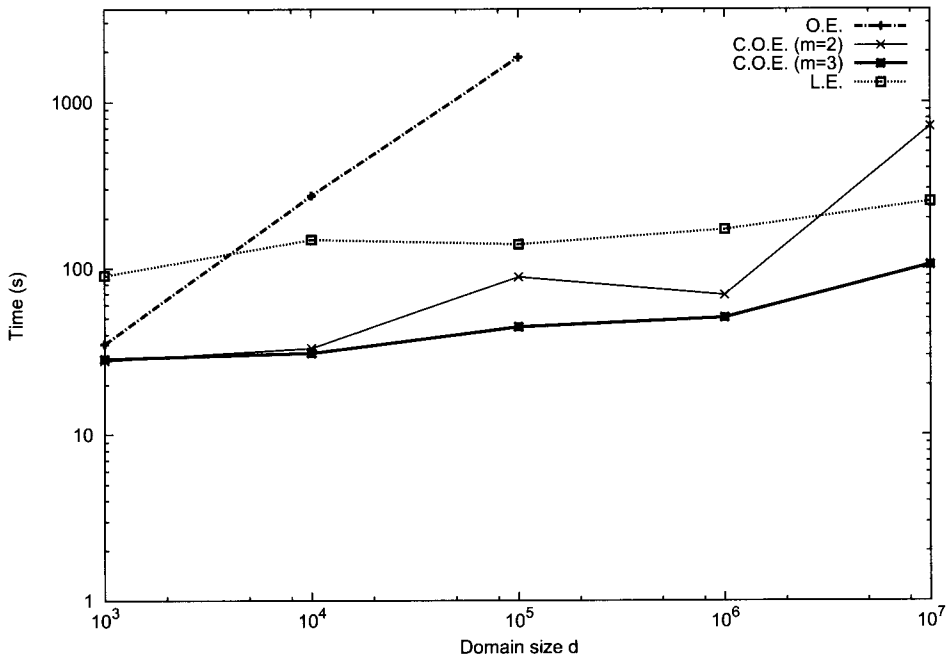

図 8.2: OSS ベンチマークにおけるドメインサイズごとの平均求解時間

表 8.3: 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

第9章　結論

本論文では以下の2つのSAT符号化の提案を行った．

- **順序符号化**
  これは，Crawfordと Bakerがジョブショップ・スケジューリング問題に適用した方法をCSPに適用可能なように一般化したものである．SATソルバーでの単位伝播が制約ソルバーの範囲伝播に対応しており，直接符号化，支持符号化，対数符号化等の他の符号化と比較して，効率の良い求解が可能である．また，各3項制約は$O(d^2)$個の節に符号化されるため，ドメインサイズが10^3以下の小規模および中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**
  ドメインサイズが10^2以下から10^7までの広い範囲の問題へ適用可能かつ効率的であることを目指して設計した方法である．コンパクト順序符号化の基本アイデアは以下の2つである．

  - 各整数変数をB進法を用いて表す$(B \geq 2)$．すなわち，各整数変数xは$m = \lceil \log_B d \rceil$桁の変数で表現される．
  - 各桁を順序符号化を用いてSATに符号化する．

  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．コンパクト順序符号化は，SATソルバーの単位伝播が最上位桁での範囲伝播に対応しているため，一般に対数符号化よりも効率が良い．また各整数を$m = \lceil \log_B d \rceil$桁で表すと，3項制約$z = x+y$と$z = xy$はそれぞれ$O(mB^2)$個と$O(mB^3+m^2B^2)$個の節に符号化され，順序符号化よりもはるかに少なくなるため，ドメインサイズが10^7程度の大規模な問題に対しても適用可能である．

グラフ彩色問題，順序符号化を採用したSAT型制約ソルバーSugarの2008年度の国際CSPソルバー競技会の結果，釣合い型不完備ブロック計画構成問題による実験結果等により，最先端の制約ソルバーと比較しても順序符号化を採用したSAT型制約ソルバーが有効であることを示した．

また，コンパクト順序符号化の有効性を検証するため，様々なドメインサイズの問題による性能評価を行い，以下の結果が得られた．

- コンパクト順序符号化が，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対して適用可能であることを確認した．
- 順序符号化や既存の SAT 符号化，最先端の制約ソルバーである Mistral と choco がほとんど解くことのできない大規模な問題であっても，コンパクト順序符号化は平均的に高速な求解を行えることを確認した．

最後に，今回提案した順序符号化およびコンパクト順序符号化は既存の SAT 符号化の欠点を解消したものであり，これにより SAT 型システムの適用範囲をさらに広げることができると考えている．

謝辞

本論文は筆者が神戸大学大学院 工学研究科 情報知能学専攻 博士課程後期課程に在籍中の研究成果をまとめたものである．同大学情報基盤センター 田村直之教授と同センター 番原睦則准教授からは，本研究の過程において数々の御助言を賜り，また粘り強く御指導をして頂いた．ここに深謝の意を表する．

また同大学 桔梗宏孝教授と玉置久教授は本論文の副査を引き受けていただき，本論文の審査過程において数々の御助言を賜った．ここに深謝の意を表する．

本研究の過程において，CSPSAT プロジェクト[1] の皆様からは多くの御助言や御激励を賜った．ここに深謝の意を表する．

最後に，大学院生活において筆者を支えてくれた神戸大学 CS18 および CS32 研究室の皆様に対し感謝の意を表する．

[1] http://www.edu.kobe-u.ac.jp/istc-tamlab/cspsat/

参考文献

[1] Olivier Bailleux and Yacine Boufkhad. 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*, pp. 108–122, 2003.

[2] Roman Barták. On-line guide to constraint programming. `http://kti.ms.mff.cuni.cz/~bartak/constraints/`, 1998.

[3] Bernhard Beckert, Reiner Hähnle, and Felip Manyà. Transformations between signed and classical clause logic. In Proceedings of the 29th International Symposium on Multiple-Valued Logics (ISMVL'99, pp. 248–255. IEEE Press, 1999.

[4] Ramón Béjar, Reiner Hähnle, and Felip Manyà. A modular reduction of regular logic to classical logic. In Proceedings of the 31st International Symposium on Multiple-valued Logic (ISMVL ' 01, pp. 221–226. IEEE Press, 2001.

[5] Christian Bessière, Emmanuel Hebrard, and Toby Walsh. Local consistencies in SAT. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003), LNCS 2919*, pp. 299–314, 2003.

[6] Armin Biere. Bounded model checking. In *Handbook of Satisfiability*, pp. 457–481. IOS Press, 2009.

[7] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. 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*, pp. 193–207, 1999.

[8] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, Vol. 185 of *Frontiers in Artificial Intelligence and Applications (FAIA)*. IOS Press, 2009.

[9] Lucas Bordeaux, Youssef Hamadi, and Lintao Zhang. Propositional satisfiability and constraint programming: A comparative survey. *ACM Computing Surveys*, Vol. 38, No. 4, 2006.

[10] Peter Brucker, Johann Hurink, Bernd Jurisch, and Birgit Wöstmann. A branch & bound algorithm for the open-shop problem. *Discrete Applied Mathematics*, Vol. 76, No. 1–3, pp. 43–59, 1997.

[11] Michael Codish and Moshe Zazon-Ivry. Pairwise cardinality networks. In *Proceedings of the 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-16)*, Vol. 6355 of *Lecture Notes in Computer Science*, pp. 154–172. Springer, 2010.

[12] Philippe Codognet and Daniel Diaz. Compiling constraints in clp(FD). *Journal of Logic Programming*, Vol. 27, No. 3, pp. 185–226, 1996.

[13] Charles J. Colbourn and Jeffrey H. Dinitz. *Handbook of Combinatorial Designs*. Chapman & Hall/CRC, 2007.

[14] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC 1971)*, pp. 151–158, 1971.

[15] James M. Crawford and Andrew B. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI 1994)*, pp. 1092–1097, 1994.

[16] Adnan Darwiche and Knot Pipatsrisawat. Complete algorithms. In *Handbook of Satisfiability*, pp. 99–130. 2009.

[17] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Communications of the ACM*, Vol. 5, No. 7, pp. 394–397, 1962.

[18] Johan de Kleer. A comparison of ATMS and CSP techniques. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI 1989)*, pp. 290–296, 1989.

[19] Maria J. García de la Banda, Kim Marriott, Reza Rafeh, and Mark Wallace. The modelling language Zinc. In *Proceedings of the 12th International Joint*

Conference on Principles and Practice of Constraint Programming (CP 2006), LNCS 4204, pp. 700–705, 2006.

[20] Rolf Drechsler, Tommi A. Junttila, and Ilkka Niemelä. Non-clausal SAT and ATPG. In *Handbook of Satisfiability*, pp. 655–693. 2009.

[21] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003), LNCS 2919*, pp. 502–518, 2003.

[22] Niklas Eén and Niklas Sörensson. Translating pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, Vol. 2, No. 1-4, pp. 1–26, 2006.

[23] Pierre Flener, Alan M. Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel, Justin Pearson, and Toby Walsh. Breaking row and column symmetries in matrix models. In *Proceedings of the 8th International Joint Conference on Principles and Practice of Constraint Programming (CP 2002), LNCS 2470*, pp. 462–476, 2002.

[24] Alan M. Frisch, Christopher Jefferson, and Ian Miguel. Symmetry breaking as a prelude to implied constraints: A constraint modelling pattern. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence (ECAI'2004)*, pp. 171–175, 2004.

[25] Allen Van Gelder. Another look at graph coloring via propositional satisfiability. *Discrete Applied Mathematics*, Vol. 156, No. 2, pp. 230–243, 2008.

[26] I. P. Gent and P. Nightingale. A new encoding of alldifferent into SAT. In *Proceedings of the 3rd International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, 2004.

[27] Ian P. Gent. Arc consistency in SAT. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI 2002)*, pp. 121–125, 2002.

[28] Ian P. Gent, Ian Miguel, and Peter Nightingale. Generalised arc consistency for the alldifferent constraint: An empirical survey. *Artificial Intelligence*, Vol. 172, No. 18, pp. 1973–2000, 2008.

[29] Carla P. Gomes, Bart Selman, and Henry A. Kautz. Boosting combinatorial search through randomization. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI 1998)*, pp. 431–437, 1998.

[30] Andrew Grayland, Ian Miguel, and Colva M. Roney-Dougal. Snake lex: An alternative to double lex. pp. 391–399, 2009.

[31] Reiner Hähnle. Uniform notation of tableau rules for multiple-valued logics. In *ISMVL*, pp. 238–245, 1991.

[32] Emmanuel Hebrard. Mistral, a constraint satisfaction library. In *Proceedings of the 3rd International CSP Solver Competition*, pp. 31–39, 2008.

[33] Jinbo Huang. Universal Booleanization of constraint models. In *Proceedings of the 14th International Joint Conference on Principles and Practice of Constraint Programming (CP 2008), LNCS 5202*, pp. 144–158, 2008.

[34] Katsumi Inoue, Takehide Soh, Seiji Ueda, Yoshito Sasaura, Mutsunori Banbara, and Naoyuki Tamura. A competitive and cooperative approach to propositional satisfiability. *Discrete Applied Mathematics*, Vol. 154, No. 16, pp. 2291–2306, 2006.

[35] 井上克巳, 田村直之. 特集「最近の SAT 技術の発展」, 第 25 巻, pp. 56–129. 社団法人人工知能学会, 2010.

[36] Kazuo Iwama and Shuichi Miyazaki. SAT-variable complexity of hard combinatorial problems. In *Proceedings of the IFIP 13th World Computer Congress*, pp. 253–258, 1994.

[37] Simon Kasif. On the parallel complexity of discrete relaxation in constraint satisfaction networks. *Artificial Intelligence*, Vol. 45, No. 3, pp. 275–286, 1990.

[38] Henry Kautz, Bart Selman, and Jöerg Hoffmann. SatPlan: Planning as satisfiability. In *Proceedings of the 5th International Planning Competition (IPC-5)*, 2006.

[39] Daniel Kroening. Software verification. In *Handbook of Satisfiability*, pp. 505–532. 2009.

[40] Daniel Le Berre and Ines Lynce. CSP2SAT4J: A simple CSP to SAT translator. In *Proceedings of the second CSP Competition*, pp. 43–54, 2008.

[41] João P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, Vol. 48, No. 5, pp. 506–521, 1999.

[42] Pedro Meseguer and Carme Torras. Exploiting symmetries within constraint satisfaction search. *Artificial Intelligence*, Vol. 129, No. 1-2, pp. 133–163, 2001.

[43] Amit Metodi, Michael Codish, Vitaly Lagoon, and Peter J. Stuckey. Boolean equi-propagation for optimized SAT encoding. In *Proceedings of the 17th International Joint Conference on Principles and Practice of Constraint Programming (CP 2011), LNCS 6876*, pp. 621–636, 2011.

[44] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC 2001)*, pp. 530–535, 2001.

[45] 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.

[46] 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.

[47] Hidetomo Nabeshima, Takehide Soh, Katsumi Inoue, and Koji Iwanuma. Lemma reusing for SAT based planning and scheduling. In *Proceedings of the International Conference on Automated Planning and Scheduling 2006 (ICAPS 2006)*, pp. 103–112, 2006.

[48] Kei Ohmura and Kazunori Ueda. 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*, pp. 524–537, 2009.

[49] Eoin O'Mahony, Emmanuel Hebrard, Alan Holland, Conor Nugent, and Barry O'Sullivan. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Proceedings of the 19th Irish Conference on Artificial Intelligence and Cognitive Science*, 2008.

[50] Justyna Petke and Peter Jeavons. 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, pp. 371–372, 2011.

[51] Steven Prestwich. Balanced incomplete block design as satisfiability. In *Proceedings of the 12th Irish Conference on Artificial Intelligence and Cognitive Science (AICS'01)*, pp. 189–198, 2001.

[52] Steven David Prestwich. Local search on SAT-encoded colouring problem. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003), LNCS 2919*, pp. 105–119, 2003.

[53] Steven David Prestwich. *Trends in Constraint Programming*, chapter 15: Finding Large Cliques Using SAT Local Search, pp. 269–274. ISTE, 2007.

[54] Steven David Prestwich. CNF encodings. In *Handbook of Satisfiability*, pp. 75–97. IOS Press, 2009.

[55] Jussi Rintanen. Planning and SAT. In *Handbook of Satisfiability*, pp. 483–504. IOS Press, 2009.

[56] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Elsevier Science Inc., New York, NY, USA, 2006.

[57] Marti Sanchez, Sylvain Bouveret, Simon de Givry, Federico Heras, Philippe Jegou, Javier Larrosa, Samba Ndiaye, Emma Rollon, Thomas Schiex, Cyril Terrioux, Gerard Verfaillie, and Matthias Zytnicki. Max-CSP competition 2008: toulbar2 solver description. In *Proceedings of the 3rd International CSP Solver Competition*, pp. 63–70, 2008.

[58] João P. Marques Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In *Handbook of Satisfiability*, pp. 131–153. 2009.

[59] Carsten Sinz. Towards an optimal CNF encoding of Boolean cardinality constraints. In *Proceedings of the 11th International Joint Conference on Principles and Practice of Constraint Programming (CP 2005), LNCS 3709*, pp. 827–831, 2005.

[60] Takehide Soh, Katsumi Inoue, Mutsunori Banbara, and Naoyuki Tamura. Experimental results for solving job-shop scheduling problems with multiple SAT solvers. In *Proceedings of the 1st International Workshop on Distributed and Speculative Constraint Processing (DSCP 2005)*, 2005.

[61] D. Tadesse, D. Sheffield, E. Lenge, R. I. Bahar, and J. Grodstein. Accurate timing analysis using SAT and pattern-dependent delay models. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '07, pp. 1018–1023, San Jose, CA, USA, 2007. EDA Consortium.

[62] Naoyuki Tamura and Mutsunori Banbara. Sugar: a CSP to SAT translator based on order encoding. In *Proceedings of the 2nd International CSP Solver Competition*, pp. 65–69, 2008.

[63] Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. 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*, pp. 590–603, 2006.

[64] Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. Compiling finite linear CSP into SAT. *Constraints*, Vol. 14, No. 2, pp. 254–272, 2009.

[65] Naoyuki Tamura, Tomoya Tanjo, and Mutsunori Banbara. System description of a SAT-based CSP solver Sugar. In *Proceedings of the 3rd International CSP Solver Competition*, pp. 71–75, 2008.

[66] 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.

[67] Tomoya Tanjo, Naoyuki Tamura, and Mutsunori Banbara. Sugar++: a SAT-based Max-CSP/COP solver. In *Proceedings of the 3rd International CSP Solver Competition*, pp. 77–82, 2008.

[68] Tomoya Tanjo, Naoyuki Tamura, and Mutsunori Banbara. Towards a compact and efficient SAT-encoding of finite linear CSP. In *Proceedings of the 9th International Workshop on Constraint Modelling and Reformulation (ModRef 2010)*, 2010.

[69] Tomoya Tanjo, Naoyuki Tamura, and Mutsunori Banbara. 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*, pp. 375–376, 2011.

[70] Tomoya Tanjo, Naoyuki Tamura, and Mutsunori Banbara. 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 (CSPSAT 2011)*, 2011.

[71] 丹生智也, 田村直之, 番原睦則. 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. コンピュータソフトウェア, 投稿中.

[72] The choco team. choco: an open source Java constraint programming library. In *Proceedings of the 3rd International CSP Solver Competition*, pp. 7–13, 2008.

[73] 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, pp. 24–35, 2010.

[74] M. R. C. van Dongen, Christophe Lecoutre, and Olivier Roussel, editors. *Proceedings of the Second International CSP Solver Competition*, 2008.

[75] Pascal van Hentenryck. *The OPL Optimization Programming Language*. MIT Press, 1999.

[76] Willem Jan van Hoeve. The alldifferent constraint: A survey. In *Proceedings of the 6th Annual Workshop of the ERCIM Working Group on Constraints*, 2001.

[77] Toby Walsh. SAT v CSP. In *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming (CP 2000)*, pp. 441–456, 2000.