論　文

基本対称関数に基づく節をもつCNF論理式の充足可能性判定

馬野　洋平[†*]　酒井　正彦[†]　西田　直樹[†]　坂部　俊樹[†]
草刈圭一朗[†]

Solving Satisfiability of CNF Formulas with Clauses Based on Elementary Symmetric Functions

Yohei UMANO[†*], Masahiko SAKAI[†], Naoki NISHIDA[†], Toshiki SAKABE[†], and Keiichirou KUSAKARI[†]

あらまし　近年，高速な充足可能性判定ツール（SAT ソルバ）の開発が進んでいる．これらのツールでは，BCP と呼ばれる変数値の推論とそのバックトラックが実行時間の大半を占めていることが多く，その処理を効率化できれば SAT ソルバの高速化が可能となる．本論文では，「n 個の変数のうち，ちょうど k 個が真」という基本対称関数に基づく節を導入することにより入力する論理式の大きさが減少することに注目し，SAT ソルバの効率化を目指す．実際に SAT ソルバでよく用いられる DPLL アルゴリズムを，基本対称関数に基づく節を導入した CNF を扱えるよう拡張し，その有効性を実験により確かめた．

キーワード　SAT ソルバ，充足可能性，基本対称関数，高速化

1. まえがき

論理式の充足可能性判定問題（SAT 問題）は NP 困難であり，計算に大変時間が掛かる厄介な問題であることが知られている．それにもかかわらず，この問題を解くための SAT ソルバ [15], [16] と呼ばれるツールの開発が近年競って進められており，実用的な時間でもかなりの問題が解けるようになってきた．種々の解きたい組合せ問題をいったん SAT 問題に変換した後に SAT ツールを利用することで，前者の問題を解くという応用があり，前者の問題を直接解くプログラムを作成するよりも高速に解けることも多いことから，この方法は有望視されている．しかしながら，非常に時間のかかり事実上解けない SAT 問題も多いため [15]，ますます SAT ソルバの高速化が求められるようになってきている．

近年実用化されてきている高速な SAT ソルバのほとんどは，論理式を和積形論理式（CNF）で入力し，DPLL アルゴリズム [3], [15] を基本アルゴリズムとして用いており，これにデータ構造の工夫や種々のヒューリスティクスの導入により高速化を実現している．DPLL アルゴリズムは，BCP（Boolean Constraint Propagation）と呼ばれる変数間の制約に基づいて論理値を伝搬させる手法を導入した DL アルゴリズム [3] にバックトラック手法を導入して得られたものである．これらのソルバにおいても，BCP 処理と BCP で伝搬された論理値のバックトラックに大半の処理時間が費される [8], [15] ため，この部分の効率化を行うことで全体の処理の更なる高速化が可能と考えられる．

本論文では，SAT ソルバへの入力形式である CNF の節（OR 節）に加えて，n 個の入力のうち k 個だけが真であるとき，かつそのときに限り，真を返す関数である基本対称関数に基づく，新しい節（ES_k 節，あるいは単に ES 節という）を導入する．複数の OR 節を一つの ES 節にまとめることで論理式を簡潔に表現でき，また，それを扱えるように BCP 手法を改良することによって，SAT ソルバを高速化する方法を提案する．これが SAT ソルバの高速化に有効であるのは，以下の理由による．

（1）　意味のある問題を解くために生成した CNF 論理式には，ES 節にまとめられる OR 節を少なから

[†] 名古屋大学大学院情報科学研究科，名古屋市
Graduate School of Information Science, Nagoya University, Furo-cho, Chikusa-ku, Nagoya-shi, 464–8603 Japan
[*] 現在，日立ソフトウェアエンジニアリング

ず含んでいること．

（2） 一つの ES_1 節により，${}_nC_2 + 1$ 個の OR 節を置き換えられること．

（3） ES 節における BCP 手法は，一般の節に対する BCP 手法とほぼ同じ手間で済むこと．

（4） 複数の OR 節を一つの ES 節にまとめるアルゴリズムは，SAT ソルバ本体の処理と比較して高速処理が可能なこと．

実際に，比較的単純な SAT ソルバの上に，ES_1 節に対する拡張を行って実装した．その際に，BCP 処理のほかに，非順序バックトラックのための影響グラフ（Implication Graph）処理の改良も必要であった．このツールを利用して，いくつかのパズル問題を解く論理式を用いた実験を行った結果，節数が 1000 分の 1 に削減できるものもあり，それらの場合には最も高速な SAT ソルバの一つである miniSAT [5] と比較しても処理時間を数分の 1 に削減できるという結果が得られた．

2. 準　　備

2.1 論 理 式

X を変数の集合とする．X 中の変数，否定記号 $\neg$，論理和 $\vee$，論理積 $\wedge$ を用いて構成される式を論理式と呼ぶ．変数割当 $\sigma : X \to \{$ 真, 偽 $\}$ に対する論理式 e の解釈結果を $\sigma(e)$ で表すことにする．論理式 e に対して，$\sigma(e)$ を真にする変数割当 σ が存在するとき e は充足可能であるといい，そうでないとき充足不能という．与えられた論理式が充足可能であるかを判定するツールのことを充足可能性判定ツール（SAT ソルバ）という．

変数 x 並びに変数の否定 $\neg x$ をリテラルという．特に x を正リテラル，$\neg x$ を負リテラルという．リテラル l に対して $l = x$ のとき $\neg x$ を，$l = \neg x$ のとき x を l の逆リテラルと呼ぶ．リテラル $l_1, \ldots, l_n$ の論理和 $l_1 \vee \cdots \vee l_n$ を OR 節という．OR 節はリテラルの集合で表すことがある．

OR 節 $C_1, \ldots, C_n$ の論理積 $C_1 \wedge \cdots \wedge C_n$ は和積形論理式または和積標準形（CNF）と呼ばれる．任意の論理式は充足可能性が等価な和積形論理式が存在する [10]．

2.2 DPLL アルゴリズム

多くの SAT ソルバに採用されている DPLL アルゴリズム [3] は，図 1 のように擬似コードで表される [15]．入力の論理式中に出現する変数のみを考える．

```
DPLL() {
    while (1) {
      if (decide_next_branch()) {
        while (dedude()==CONFLICT) {
          blevel = analyze_conflicts();
          if (blevel < 0)
            return UNSATISFIABLE;
          else
            back_track(blevel);
        }
      }
      else
        return SATISFIABLE;
    }
}
```

図 1　SAT ソルバの DPLL アルゴリズム
Fig. 1　DPLL algorithm for SAT solvers.

このアルゴリズム中では，各変数は真か偽の値が割り当てられている状態か，あるいは，値が割り当てられていない状態をもっている．値が割り当てられていない変数を未定変数と呼び，未定変数をもつリテラルを未定リテラルと呼ぶ．また，真（偽）であるリテラルを真（偽）リテラルと呼ぶ．

アルゴリズム開始時点ではすべての変数は未定変数であり，変数に値が割り当てられた順番を管理するために空スタックを用いる．スタックには変数のほかにマーク情報も一緒に入れられ，スタック中のマークの数を**レベル**と呼び，プログラム変数 `blevel` で管理する．

アルゴリズムでは，以下の処理を繰り返し実行する．

（1） `decide_next_branch()` によって未定変数 x を一つ選んで値を真か偽のいずれかに割り当てて，マークを付けて x をスタックに入れる．このとき，未定変数がなければ e は充足可能とし停止する．

（2） `deduce()` によって BCP 処理，すなわち，OR 節を真にするために自動的に値が定まる変数の推論を繰り返し行う．これによって値が割り当てられた変数はマークせずにスタックに入れる．

（3）（2）の推論の際に，**衝突**，すなわち，ある OR 節が偽と分かった場合には，`analyze_conflicts()` により，どのマークされた変数までバックトラックすべきかを調べる(注1)．もしスタックが空になるまで戻る

(注1)：マークされた変数のうちで最後にスタックに入れられた変数まで戻るのが，最も単純な戦略である．

必要がある場合には，充足不能とし停止する．

（4） `back_track()` により，（3）でマークされた変数 x まで値を未定に戻しスタックから除去する．変数 x については，値を反転させてマークせずにスタックにいれて，（2）に戻る．

（5）（2）の推論で衝突が起こらなかった場合には，（1）に戻る．

以下では，DPLL アルゴリズムを本論文での議論に必要な部分，すなわち入力の CNF を直接参照する部分について，より詳しく述べる．

2.2.1 BCP 処理

ここでは，先に述べた `deduce()` における，変数の推論と衝突の検出について説明を加える．論理式は CNF 形式で与えられているため，論理式を真とするために未定変数の値が定まる場合は，一つのリテラルのみが未定変数からなり，かつ，その他のリテラルが偽である OR 節が存在する場合のみである．その際には，未定リテラルが真になるように未定変数の値が推論される．

また，すべてのリテラルが偽であるような OR 節が存在する場合には，論理式全体が偽となるため，これを衝突として取り扱う．

これらの条件をまとめると表 1 のようになる．

2.2.2 影響グラフ

`analyze_conflicts()` によるバックトラックの戻り先の解析において，先の BCP 処理の履歴を保存しておくことにより，衝突に無関係なマーク変数をも飛び越したバックトラックが可能である（非順序バックトラック）[7], [15]．この BCP 処理の履歴は影響グラフ（Implication Graph）と呼ばれる．また，この情報は，高速化のための衝突からの OR 節の学習にも利用される [15]．

影響グラフの頂点はリテラルとレベルの組で構成され，BCP 処理が進むたびに頂点と矢が追加される．偽リテラル $l_1, \ldots, l_p$ と未定リテラル l により推論が行われたとする．このとき，

表 1　未定変数値の推論と衝突検知の条件
Table 1　Conditions of constraint propagation and collision detection.

条件	動作
$\#_f = n-1$ かつ $\#_t = 0$	未定リテラルを真とする推論
$\#_f = n$	衝突検知

n：節に含まれるリテラルの数
$\#_f$：偽リテラルの数

（1） 頂点 $\langle l, d\rangle$ を追加する．

（2） 各 i（$1 \le i \le p$）について，l_i の逆リテラルをもつ頂点から $\langle l, d\rangle$ への矢を加える．

ここで，d は処理が行われた時点でのレベルである．

また，バックトラックの際には，その処理後のレベル以上のレベルをもつ頂点とそれに関連する矢を削除する．

非順序バックトラックや学習がうまく働くためには，影響グラフ $G = \langle V, E\rangle$ が常に以下の性質を満たしている必要がある．

（1） 変数 x がレベル d で真に割り当てられているとき，かつそのときに限り，$\langle x, d\rangle \in V$ である．

（2） 変数 x がレベル d で偽に割り当てられているとき，かつそのときに限り，$\langle \neg x, d\rangle \in V$ である．

（3） BCP 処理で節 C を用いた推論により追加された各頂点 $\langle l, d\rangle$ について，それに向かう矢の始点が $\langle l_1, d_1\rangle, \ldots, \langle l_m, d_m\rangle$ とする．このとき，$l_1, \ldots, l_m$ をすべて真とし，かつ，l を偽とする変数割当のもとで，C は偽である．

3. ES 節の導入

3.1 基本対称関数に基づく節をもつ CNF

論理関数 $f(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n)$ において，任意の変数 x_i と x_j の対を入れ換えて得られる関数が元の関数に等しいとき，f を対称関数（symmetric function）という．また，これらのうちで，n 個の変数のうち，ちょうど k 個が真であるときのみ f の関数値が真となる n 変数論理関数 S_k^n $(0 \le k \le n)$ を n 変数基本対称関数（elementary symmetric function）という．任意の対称関数は，いくつかの基本対称関数の論理和で表されることが知られている．例えば n 引数 OR 関数はその中のリテラルの順番を入れ換えても式の値は変わらないため対称関数であり，$\mathrm{OR}(x_1, \ldots, x_n) = S_1^n(x_1, \ldots, x_n) \vee \cdots \vee S_n^n(x_1, \ldots, x_n)$ と表せる．

基本対称関数 S_k^n に基づいて，ES 節を，以下のように定義する．ES_k 節は，k 個以上のリテラルからなるリテラルの集合である．リテラルの個数を n とするとき，その ES_k 節は，それらのリテラルの値を基本対称関数 S_k^n の引数に与えて得られる値と解釈する．例えば，ES_1 節 $\{x_1, x_2, \neg x_3\}$ は，論理式 $S_1^3(x_1, x_2, \neg x_3)$ と解釈される，すなわち，リテラル $x_1, x_2, \neg x_3$ のうちいずれか 1 個が真で残りの 2 個が偽であるとき，かつそのときに限り，真となる．

OR 節と ES_k 節の両方をまとめて節と呼ぶ．K を

自然数の有限集合とするとき，いくつかの OR 節と ES_k 節（$k \in K$）の論理積を基本対称関数に基づく節をもつ CNF と呼び，以下では単に ES_KCNF と書く．

3.2 OR 節と ES 節の関係

ES 節は，複数個の OR 節を用いて表すことができる．例えば，ES_1 節 $\{x_1, x_2, \neg x_3\}$ は，4 個の OR 節 $\{x_1, x_2, \neg x_3\}$，$\{\neg x_1, \neg x_2\}$，$\{\neg x_1, x_3\}$，$\{\neg x_2, x_3\}$ で表せる．これを n 個のリテラルをもつ ES_1 節 $\{l_1, \ldots, l_n\}$ に一般化すると，次のように表せる．

$$\left(\bigvee_{k=1}^{n} l_k\right) \wedge \bigwedge_{i=1}^{n} \bigwedge_{j=i+1}^{n} (\neg l_i \vee \neg l_j)$$

一般に，n 個のリテラルからなる ES_k 節 $\{l_1, l_2, \ldots, l_n\}$ は，次の等価な CNF に変換できる．

$$\left(\bigwedge_{t_1=1}^{n} \bigwedge_{t_2=t_1+1}^{n} \cdots \bigwedge_{t_{n-k+1}=t_{n-k}+1}^{n} \bigvee_{i=1}^{n-k+1} l_{t_i}\right) \wedge \left(\bigwedge_{t_1=1}^{n} \bigwedge_{t_2=t_1+1}^{n} \cdots \bigwedge_{t_{k+1}=t_k+1}^{n} \bigvee_{i=1}^{k+1} \neg l_{t_i}\right)$$

この CNF の節数は ${}_nC_{k-1} + {}_nC_{k+1}$ である．すなわち，ES_1 節で $O(n^2)$，ES_2 節で $O(n^3)$，ES_3 節で $O(n^4)$ のオーダとなる．ここでは，OR 節による ES 節の自然な表現法を述べたが，文献 [1], [9] の手法を用い，新しい変数を導入することで充足可能性を保存したまま少ない数の OR 節で ES_k 節を表現することも可能である．具体的には ES_k 節は，[1] の方法に基づけば $O(n^2)$ の OR 節で，[9] では $O(kn)$ の OR 節で表現可能である．

4. DPLL アルゴリズムの拡張

前章で述べた DPLL アルゴリズムを ES_KCNF に対しても動作させるためには，**2.2.1** と **2.2.2** で述べた入力の CNF を参照する部分，すなわち，BCP 処理と影響グラフの拡張が必要である．

4.1 BCP 処理

BCP 処理は，ES 節の意味を考えれば容易に拡張でき，表 2 に示す条件を考えればよい．

未定変数の値の推論について，リテラル数 n の ES_k 節を真にするためには，

（1）「偽リテラル数が $n-k$」ならば，他のリテラルはすべて真の必要があり，また，

（2）「真リテラル数が k」ならば，他のリテラルはすべて偽の必要がある．

表 2 ES_k 節に対する未定変数値の推論と衝突検知の条件
Table 2 Conditions of constraint propagation and collision detection for ES_k clauses.

条件	動作
$\#_f = n-k$	すべての未定リテラルを真とする推論
$\#_t = k$	すべての未定リテラルを偽とする推論
$\#_t > k$	衝突検知
$\#_f > n-k$	

n：節に含まれるリテラルの数
$\#_t$：真リテラルの数
$\#_f$：偽リテラルの数

このように，ES_k 節では，OR 節と比較してより多くの場合に推論が可能であり，また，これにより複数の未定変数の値を一度に定めることができるため，効率的である．

衝突の検出に対しては，節が偽になる条件を考えれば十分であるので，その条件は「真リテラル数が k より大きい，または，偽リテラル数が $n-k$ より大きい」となる．衝突についても，OR 節の場合よりも多くの場合に検知が可能である．

4.2 影響グラフ

影響グラフの各頂点が **2.2.2** で述べた性質を満たすためには，ES_k 節によるレベル d での BCP に対する影響グラフの更新は次のようにすればよい．

（1） 未定リテラルを真とする推論が起きたとき：偽リテラルを $l_1, \ldots, l_{n-k}$ とし，未定リテラルを $l'_1, \ldots, l'_p$ とすると，各 j（$1 \le j \le p$）に対して，

(i) 頂点 $\langle l'_j, d\rangle$ を追加する．

(ii) 各 i（$1 \le i \le n-k$）について，l_i の逆リテラルをもつ頂点から $\langle l'_j, d\rangle$ への矢を加える．

（2） 未定リテラルを偽とする推論が起きたとき：真リテラルを $l_1, \ldots, l_k$ とし，未定リテラルを $l'_1, \ldots, l'_p$ とすると，各 j（$1 \le j \le p$）に対して，

(i) 頂点 $\langle l''_j, d\rangle$ を加える．ここで l''_j は l_j の逆リテラルである．

(ii) 各 i（$1 \le i \le k$）について，l_i をもつ頂点から $\langle l''_j, d\rangle$ への矢を加える．

5. $\mathrm{ES}_{\{1\}}$-CNF に対応した SAT ソルバの実装

まず，ベースになる SAT ソルバ（nanosat_cnf）を説明する．この SAT ソルバは，picosat2 [16] をもとに，zchaff [12], [13]，quaffle [14] を参考にして高速化の機能を追加することによって作成した．具体的には，

DPLL アルゴリズム，衝突からの節の学習，非順序バックトラック，VSIDS 変数選択ヒューリスティクス，学習節の忘却の基本的な機能とそのヒューリスティクス [7], [8], [15] が実装されている．なお，VSIDS は衝突の原因となった変数の出現頻度に応じて変数選択を行うヒューリスティクスであり，変数選択のコストが少なく効率的とされる．また，BCP 処理については，それぞれの節に対して真リテラル数と偽リテラル数を保持するための二つのカウンタを用いる方法 [7], [15] を採用した．その理由は，ES_K-CNF への拡張が容易に実現できると考えたからである．

ベースとなる nanosat_cnf に対して，前章で述べた拡張を加えることにより $\mathrm{ES}_{\{1\}}$CNF の入力に対応した SAT ソルバ（nanosat）を作成した．

5.1 入力表現

SAT ソルバへの標準的な CNF の入力形式では，例えば CNF 論理式 $(x_1 \vee x_4) \wedge (x_2 \vee \neg x_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3)$ は，図 2 のように表される．この入力形式では，変数は 1 から始まる自然数で表され，変数 n の負リテラルは-n で表現される．先頭行の `p cnf <vars> <clauses>`の形式は，`<vars>`で出現する最大の変数を，`<clauses>` で節数を表している．2 行目以降には，1 行当り 1 つの節を表し，各節はリテラルを空白で区切って並べ，最後に `0` を置くことになっている．なお，`c` で始まる行はコメントとして解釈される．

この入力表現を拡張し，先頭に ! が付けられた節を，ES_1 節と定義した．先の CNF 論理式は，二つの OR 節 $\{x_1, x_4\}$，$\{x_2, \neg x_4\}$ と一つの ES_1 節 $\{x_1, x_2, x_3\}$ で表すことができ，図 3 のように表現できる．

5.2 CNF から $\mathrm{ES}_{\{1\}}$CNF への変換

CNF 形式で記述された入力に対しても，$\mathrm{ES}_{\{1\}}$CNF 形式に変換してから問題を解くことを可能にするため，入力から ES_1 節を抽出するモジュールも実装した．

CNF から $\mathrm{ES}_{\{1\}}$CNF への変換は，**3.2** で述べた OR 節と ES_1 節の関係に基づき，入力中の各 OR 節 $\{l_1, \ldots, l_n\}$ について次のことを調べればよい．

(*) $1 \le i < j \le n$ を満たすすべての i，j について OR 節 $\neg l_i \vee \neg l_j$ が CNF 内に存在する．

これを満たしている場合には，それらの $\neg l_i \vee \neg l_j$ の ${}_nC_2$ 個の OR 節，並びに，OR 節 $\{l_1, \ldots, l_n\}$ を，ES_1 節 $\{l_1, \ldots, l_n\}$ に置換する．

本システムの実装では，以下のデータを準備してから前述の (*) を調べるようにしている．

```
p cnf 4 5
1 4 0
2 -4 0
1 2 3 0
-1 -2 0
-1 -3 0
-2 -3 0
```

図 2　CNF の入力表現
Fig. 2　Input representation of a CNF.

```
p escnf 4 3
1 4 0
2 -4 0
! 1 2 3 0
```

図 3　$\mathrm{ES}_{\{1\}}$CNF の入力表現
Fig. 3　Input representation of a $\mathrm{ES}_{\{1\}}$CNF.

（1）　二つのリテラルの関係 R を表す構造を用意する．初期値は $R = \emptyset$ とする．

（2）　入力中の長さ 2 の各 OR 節 $\{l_1, l_2\}$ $(l_1 < l_2)$ について，(l_1, l_2) を R に追加する．

節の数を m，変数の数を n とすると，本システムでは R のデータ構造として，リストの配列を用いているため，データの追加並びに検索には $O(n)$ かかる．よって，上記の関係 R の構築に $O(m \times n)$ かかる．また，m 個の節のそれぞれに対して (*) の判定に $O(n^3)$ かかり，合計で $O(m \times n^3)$ の最大計算量となる．

本システムではこの部分の効率化はあまり考えずに実装しているが，リテラル間に順序をつけて，各節や関係 R のデータをソートすることにより，オーダ $O(m \times n^2)$ の実装が可能である．

6. 調査・実験結果

本章では，まず実用上の問題にどの程度 ES_1 節が含まれるのかについて調査結果を述べる．次に，CNF 論理式をベースシステム nanosat_cnf で解く場合と，その論理式を $\mathrm{ES}_{\{1\}}$CNF 論理式に変換の後に nanosat で解く場合に必要な実行時間について比較実験を行った結果について述べる．

6.1 $\mathrm{ES}_{\{1\}}$CNF への変換による節の減少度

実用問題において，**5.2** で述べた変換による節数の削減割合について調査した．調査の対象とした問題は，

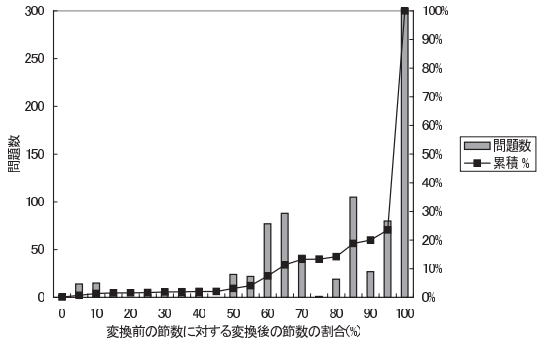

図 4　節の削減比率の分布
Fig. 4　Distribution of reduction ratio of clauses.

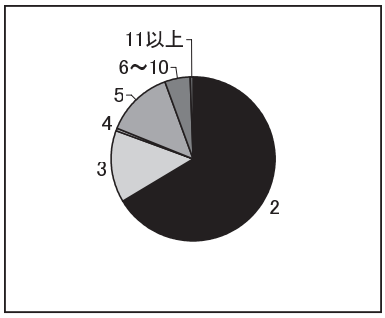

図 5　ES_1 節数の ES_1 節の長さによる分類
Fig. 5　Classification of ES_1-clauses by their length.

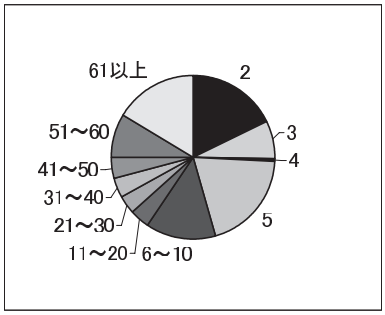

図 6　削除された OR 節数の ES_1 節の長さによる分類
Fig. 6　Classification of OR-clauses by the length of produced ES_1-clauses.

表 3　SAT Competition 問題での解けた問題数と演算時間
Table 3　Number of solved problems and execution time for SAT-competition problems.

ソルバ	問題群 A		問題群 B	
	解けた数	演算時間	解けた数	演算時間
本手法	**21**	**3438 秒**	**42**	**9482 秒**
従来手法	17	3517 秒	40	9588 秒
（参考値）	(19)	(3447 秒)	(51)	(9127 秒)

SAT ソルバの競技会である SAT Competition [17] で使用された問題のうち，2003 年から 2007 年までの Industrial 部門と Crafted 部門の問題 1915 個を用いた．なお，2004 年の Industrial 部門についてはデータの入手ができなかったため，また，2006 年には競技会が開催されていないため，これらの問題は対象に含まれていない．

集計結果を図 4 に示す．この表では，変換後の節数の，変換前の節数に対する割合を横軸にとり，その変換割合に該当する問題数を棒グラフで，その変換割合以下に該当する問題の累積割合を折れ線グラフで表してある．このグラフから，調査した問題のうち 20%の問題で，節数が 90%以下に，すなわち，10%以上の数の節が削減される，また，10%の問題で節数が 60%以下に，すなわち，40%以上の数の節が削減されることが分かる．このように実用上の問題でも $ES_{\{1\}}$CNF で表現すると節数が減少する問題が少なくないことが確認できた．

また，これらの 1915 問の問題に対して，変換後に得られた ES_1 節の長さ n について特徴を調べた．図 5 は，変換で得られた ES_1 節の出現数の比率を n で分類したものであり，図 6 は，ES_1 節への変換で削除された OR 節の数を n で分類したものである．これによると，得られた ES_1 節のほとんどは長さ 5 以下であるが，削除された OR 節の個数，すなわち，節数の削減に対する貢献度で考えると，長さ 11 以上の ES_1 節もかなり多いことが分かった．

6.2　SAT Competition 問題に対する速度比較

本節では，SAT Competition 問題を用いて本論文の手法を評価する．

比較結果を表 3 に示す．ここで，前節で調査した SAT Competition の問題のうち，節数の比が 60%以下の 133 問を「問題群 A」とし，90%以下の 349 個を「問題群 B」とした．

評価に用いたソルバは以下のとおりである．

- **5.** で述べた nanosat で問題を $ES_{\{1\}}$CNF に変換してから解く（本手法）
- nanosat のベースシステムである nanosat_cnf で解く（従来手法）
- 高速性に定評のある minisat 1.14 [5] で解く（参考値）

なお，これらのソフトウェアは cygwin gcc 4.3.2 を用いてコンパイルし，CPU Intel E8200（2.66 GHz × 2），メモリ 2 GByte のハードウェア上で処理時間を

測定した．表中の「処理時間」には対象の問題群の処理にかかったすべての時間が，一問当りのタイムアウトの時間 30 秒も含めて加算してある．また，nanosat については，CNF から ESCNF への変換時間（問題群 A：113 秒，問題群 B：341 秒）も加算された値である．

本手法を従来手法と比較すると，節数比がいずれの場合も，より多くの問題を解くことができるようになり，演算時間は高速になった．特に，$ES_{\{1\}}$ECF への変換の際に節数の減少率が大きかった問題群 A については，本手法で解けた問題数が minisat による場合よりも上回る結果となった．

問題群 A に対しては，従来手法で解けた 17 問のすべてが本手法により解けた．これらのうち充足不能であったのは 2 問であり，本手法のみで解けた問題 4 問はすべて充足可能であった．これに対して問題群 B に対しては，従来手法と本手法の両方で解けた問題数は 35 問であり解けた問題が一部異なっていたが，充足不能な問題 4 問は両方で解けた．nanosat では変数選択ヒューリスティクスにおいて nanosat_cnf となるべく同じ変数が選ばれるようにするため，ES_1 節を OR 節で表現した場合に換算して変数の出現数を定めている．しかしながら，CNF から $ES_{\{1\}}$CNF への変換により節の出現順序を同一にすることができず，変数の評価値が同じ場合の変数選択が異なってしまい，解けた問題が一部異なったものと思われる．

このように，本手法では従来手法よりも充足可能な問題をより多く解くことができ，問題の変換時間を考慮しても，基本対称関数に基づく節をもつ CNF による SAT ソルバの高速化を確認できた．

6.3 パズル問題における速度比較

パズルを SAT 問題に変換して解く場合には，しばしばいくつかの自然数を 1 進数表現でコーディングすることもあり，本質的に ES_1 節で表されることを論理式にもつことが多い．例えば，数独 [18] 問題のコーディング [6] では，生成される論理式のほとんどの部分が ES_1 節として記述可能であるし，また，その方が直観的であり理解しやすい．

そこで，数独，魔方陣，お絵書きロジック，ナイト巡回問題，ハノイの搭のそれぞれのパズルについて，その問題を解くための論理式を利用して，本手法の評価を行った．先にも述べたようにパズル問題においては，直接 ES_1 節を生成した方が自然であるため，CNF からの変換を行わずに，CNF 形式の SAT 問題と $ES_{\{1\}}$ 形式の SAT 問題の両方を用意して実験を行った．実験結果を表 4 に示す．ここで，節数比は，各問題をコーディングした CNF 論理式の節の数に対する $ES_{\{1\}}$ 論理式の節の数の比を表す．また，計算機環境は前節と同一であり，表中の Error はメモリオーバフローにより停止したことを表している．

表 4　パズル問題の演算時間（秒）
Table 4　Execution time for puzzle problems (s).

実験データ	**nanosat（本手法）**	nano_cnf（従来法）	minisat（参考値）	節数比
数独 25 × 25	**0.08**	2.32	(0.44)	0.33%
数独 36 × 36	**0.25**	9.89	(2.23)	0.16%
数独 49 × 49	**0.81**	36.20	(7.64)	0.08%
数独 64 × 64	**3.87**	Error	31.41	0.05%
数独 81 × 81	**7.99**	Error	(Error)	0.03%
魔方陣 4 × 4	**0.16**	0.27	(0.25)	34%
魔方陣 5 × 5	1513	**513**	(Error)	30%
お絵かき 40 × 50	**0.73**	3.41	(0.19)	30%
お絵かき 50 × 60	**1.27**	2.72	(1.95)	30%
ナイト巡回 6 × 6	**0.30**	0.44	(0.31)	68%
ナイト巡回 7 × 7	**0.72**	1.03	(0.13)	69%
ナイト巡回 8 × 8	78.7	**76.6**	(16.8)	69%
ナイト巡回 9 × 9	**6.52**	7.94	(0.47)	69%
ハノイの塔 4 枚	**0.45**	0.86	(0.11)	70%

強調文字は nanosat_cnf と nanosat をうち高速な方を表す．

この実験結果から本手法がパズル問題において効果的であることが分かる．特に，サイズの大きい数独や魔方陣など，minisat では記憶領域不足のため解けなかった問題についても，nanosat では比較的短時間で解くことが可能であるなど，特徴的な結果が得られた．

6.4 オーバヘッドの評価

CNF 形式で与えられた SAT 問題をベースシステムである nanosat_cnf と，これに本手法を実装した nanosat で $ES_{\{1\}}$CNF に変換せずに解くことにより，オーバヘッドを測定した．実験は前節までと同一の条件で行った結果，オーバヘッドはおおむね 1%程度であった．例えば，SAT Competition 問題の場合には，解けた問題数に違いはなく，処理に要した時間は問題群 A で 3518 秒，問題群 B で 9591 秒と nanosat_cnf と比較して 1%以下の違いであった．

7. むすび

本論文では，「n 個のリテラルのうち，ちょうど k 個が真」を意味する ES_k 節を CNF に導入して SAT 問題を解く手法を示した．実際に，SAT ソルバアルゴリズムで用いられる BCP 処理と影響グラフへの拡張法を示し，SAT ソルバを実装した．実験の結果，この手法は実用上の問題を解くのに有効であることが確認で

きた．

本研究では 2 カウンタ法による BCP 処理法に基づく実装を行ったが，これ以外の方法，例えば，2 ポインタによる BCP 処理法に基づく実装 [8], [11] に対して，いかに効率良く拡張するかを研究することは重要な課題である．また，ES_KCNF への拡張の効果を評価することも課題として挙げられる．

本手法で与えた ES_KCNF は擬ブール論理式（Pseudo Boolean formula）の特別な式とみなせるため，それらのソルバ（例えば [2]）との比較，あるいは，擬ブール論理式への拡張も課題である．

文　献

[1] O. Bailleux and Y. Boufkhad, “Efficient CNF encoding of boolean cardinality constraints,” Proc. International Conference on Principles and Practice of Constraint Programming, LNCS 2833, pp.108–122, 2003.

[2] D. Chai and A. Kuehlmmann, “A fast pseudo-boolean constraint solver,” IEEE Trans. Comput.-Aided Des. Integr. Cicuits Syst., vol.24, no.3, pp.305–317, 2005.

[3] M. Davis, G. Logemann, and D. Loveland, “A machine program for theorem proving,” Commun. ACM, vol.5, no.7, pp.394–397, July 1962.

[4] M. Davas and H. Putnam, “A computing procedure for quantification theory,” J. ACM, vol.7, no.3, pp.201–215, July 1960.

[5] N. Eén and N. Sörensson, “An extensible SAT-solver,” Proc. Sixth International Conference on Theory and Applications of Satisfiability Testing, LNCS 2919, pp.502–518, 2004.

[6] G. Kwon and H. Jain, “Optimized CNF encoding for sudoku puzzles,” Proc. 13th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR2006), pp.1–5, 2006.

[7] J.P. Marques-Silva and K. Sakallah, “GRASP: A new search algorithm for satisfiability,” Proc. International Conference on Computer-Aided Design, pp.220–227, Santa Clara, 1996.

[8] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik, “Chaff: Engineering an efficient SAT solver,” 39th Design Automation Conference (DAC 2001), pp.530–535, Las Vegas, June 2001.

[9] C. Sinz, “Towards an optimal CNF encoding of boolean cardinality constraints,” Proc. International Conference on Principles and Practice of Constraint Programming, LNCS 3709, pp.827–831, 2005.

[10] G. Tseitin, “On the complexity of derivation in propositional calculus,” Studies in Constructive Mathematics and Mathematical Logic, pp.115–125, 1968.

[11] H. Zhang, “SATO: An efficient propositional prover,” Proc. International Conference on Automated Deduction (CADE'97), LNAI 1104, pp.308–312, 1997.

[12] L. Zhang, C.F. Madigan, M.H. Moskewicz, and S. Malik, “Efficient conflict driven learning in a boolean satisfiability solver,” Proc. International Conference on Computer Aided Design (ICCAD 2001), pp.279–285, San Jose, Nov. 2001.

[13] L. Zhang and S. Malik, “The quest for efficient boolean satisfiability solvers, invited paper,” Proc. 8th International Conference on Computer Aided Deduction (CADE 2002) and also Proc. 14th Conference on Computer Aided Verification (CAV2002), LNCS 2404, pp.17–36, July 2002.

[14] L. Zhang and S. Malik, “Conflict driven learning in a quantified boolean satisfiability solver,” Proc. International Conference on Computer Aided Design (ICCAD 2002), pp.442–449, San Jose, Nov. 2002.

[15] L. Zhang, Searching for Truth: Techniques for Satisfiability of Boolean Formulas, Ph.D. Thesis, Princeton University, 2003.

[16] L. Zhang, “SAT-Solving: From davis-putnam to zchaff and beyond,” Lecture Notes, IT University of Copenhagen, 2003.

[17] SAT Competitions. http://www.satcompetition.org

[18] ニコリ公式パズルガイド「数独」 http://www.nikoli.co.jp/ja/puzzles/sudoku/

（平成 21 年 4 月 15 日受付，8 月 20 日再受付）

馬野　洋平

2005 名大・理・数理学卒．民間企業を経て，2009 同大大学院情報科学研究科計算機数理科学専攻博士前期課程了．同年，日立ソフトウェアエンジニアリング（株）入社．充足可能性判定ツールに関する研究に従事．

酒井　正彦　（正員）

1989 名古屋大学大学院博士課程満了．同年同大工学部助手．1993 北陸先端科学技術大学院大学助教授．1997 名古屋大学工学研究科助教授．2002 同教授．2003 同大大学院情報科学研究科教授，現在に至る．この間，1996 年 3 月〜8 月ニューヨーク州立大学スートーニーブルック校客員研究教授．項書換え系などのソフトウェア基礎理論に関する研究に従事．工博．平 3 年度本会論文賞受賞．日本ソフトウェア科学会会員．

西田　直樹　（正員）

2000 名大･工･電気電子･情報工卒，2002 同大大学院工学研究科計算理工学博士前期課程了，2004 同大学院工学研究科情報工学専攻博士後期課程了，2004 同大学院情報科学研究科助手，2007 より同助教，現在に至る．項書換え系，プログラム変換に関する研究に従事．工博．日本ソフトウェア科学会会員．

坂部　俊樹　（正員）

1972 名大・工・電気卒．1977 同大大学院博士課程了．名古屋大学助手，三重大学助教授，名古屋大学助教授を経て，1993 より名古屋大学教授．ソフトウェアの基礎理論全般に興味をもつ．最近では，抽象データ型の理論，プログラミング言語の形式的意味論，自動プログラミング，書換え型計算モデル，並行プロセスの理論などの研究に従事．工博．情報処理学会，人工知能学会，日本ソフトウェア科学会，EATCS 各会員．

草刈圭一朗　（正員）

1994 東工大・理・生命理学卒．1996 北陸先端科学技術大学院大学情報科学研究科博士前期課程了．2000 同大学博士後期課程にて博士（情報科学）取得．同年東北大学電気通信研究所助手．2003 名古屋大学大学院情報科学研究科情報システム学専攻講師．2006 より同大学院研究科計算機数理科学専攻助教授．項書換え系・プログラム理論・定理自動証明の研究に従事．情報処理学会・日本ソフトウェア科学会各会員．