特　集

高速SATソルバーの原理

Principles of Modern SAT Solvers

鍋島 英知
Hidetomo Nabeshima
山梨大学大学院医学工学総合研究部
Department of Research Interdisciplinary Graduate School of Medicine and Engineering, University of Yamanashi
nabesima@yamanashi.ac.jp

宋 剛秀
Takehide Soh
総合研究大学院大学複合科学研究科情報学専攻
Department of Informatics, School of Multidisciplinary Sciences, The Graduate University for Advanced Studies
soh@nii.ac.jp, http://research.nii.ac.jp/il/soh/

keywords: SAT solvers, conflict-driven clause learning, watched literals, decision heuristics, restart strategy

1. は　じ　め　に

近年，命題論理の充足可能性判定問題（SAT 問題）を解くソルバーの性能は飛躍的に向上している．システム検証やプランニング問題，スケジューリング問題，制約充足問題，制約最適化問題，定理証明等の様々な分野において SAT を利用した問題解決手法が開発され，大きな性能の向上をもたらしている．本稿では，近年の高速 SAT ソルバーにおける代表的な要素技術を紹介する．

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，確率的に解を探索し充足可能のみを判定する**確率的 SAT ソルバー**の二種類が存在する．その中でも，実応用問題から得られる構造的な SAT 問題に対して近年大きく性能を向上させているのが，DPLL 手続き [Davis 62] に基づいた系統的 SAT ソルバーである．その背景にある高速化技術には，探索過程で矛盾が発生した際にその原因を解析・記録することで同様の矛盾の発生を防ぐ節学習とそれに基づくバックジャンプ法 [Silva 99b, Bayardo 97] や，監視リテラルによる高速な単位伝播 [Moskewicz 01]，優れた変数選択ヒューリスティックスの開発 [Moskewicz 01]，変数の選択順序による影響を緩和するためのリスタート戦略 [Gomes 98] 等がある．近年の DPLL ベースの SAT ソルバーの多くは，これらの技術を実装している．その中でも，Eén と Sörensson による MiniSat は，これらの高速化技術を非常に簡潔かつ効率良く実装することで，大きな性能向上を果たした代表的な SAT ソルバーである．

以降では，まず 2 章で高速 SAT ソルバーの基礎となっている DPLL 手続きを示し，続く 3 章で高速 SAT ソルバーの要素技術と最新の動向を紹介する．4 章では MiniSat の利用方法を簡単に示す．5 章では，関連研究として，ランダムに生成された SAT 問題に対して優れた性能を示している先読み型の SAT ソルバーの手法を紹介する．最後に 6 章で本稿をまとめる．

2. SAT 問題と DPLL 手続き

命題論理の**充足可能性判定** (Boolean / propositional satisfiability (testing); **SAT**) は，与えられた命題論理式の充足可能性または充足不能性を判定する問題である．SAT の各インスタンスを **SAT 問題** (SAT problem instance) と呼ぶ．SAT は，通常 **連言標準形** (conjunctive normal form; **CNF**) の論理式で与えられる．CNF 式は，命題変数またはその否定を表す**リテラル** (literal) の選言 ($\vee$) の連言 ($\wedge$) である．リテラルの選言を**節** (clause) と呼ぶ．本稿では命題変数を $x, x_1, x_2, \ldots$ で表す．また節をリテラルの集合で表し，CNF 式を節の集合で表すことがある．例えば CNF 式 $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3 \vee x_4)$ は $\{\{x_1, \neg x_2\}, \{x_2, \neg x_3, x_4\}\}$ と表される．リテラルを命題変数に変換する関数 *var* を，$\mathit{var}(x) = \mathit{var}(\neg x) = x$ と定義し，リテラルの符号を取り出す関数 *sign* を，$\mathit{sign}(x) = 1, \mathit{sign}(\neg x) = 0$ と定義する．命題変数は**真** (1) または**偽** (0) の値をとるが，SAT ソルバーの探索過程においては値が未割り当ての場合がある．そこで変数の**値割り当て** (assignment) を写像 $\nu : X \to \{0, 1, \mathsf{u}\}$ と定義する．ここで X は命題変数の有限集合，u は未割り当てを表す[*1]．変数の値割り当て ν を，リテラル l，節 $C = \{l_1, \ldots, l_n\}$，CNF 式 $\psi = \{C_1, \ldots, C_m\}$ の値割り当てとして以下のように拡張する．ただし $0 < \mathsf{u} < 1$ かつ $1 - \mathsf{u} = \mathsf{u}$ とする．

$$\nu(l) = \begin{cases} \nu(x) & \text{if } l = x \\ 1 - \nu(x) & \text{if } l = \neg x \end{cases}$$
$$\nu(C) = \max\{0, \nu(l_1), \ldots, \nu(l_n)\}$$
$$\nu(\psi) = \min\{1, \nu(C_1), \ldots, \nu(C_m)\}$$

値割り当て ν において式 ψ（または節 C）が**充足される**とは，$\nu(\psi) = 1$（または $\nu(C) = 1$）のときをいう．逆に $\nu(\psi) = 0$（または $\nu(C) = 0$）のとき，ψ（または C）は

∗1　[井上 10] では，真偽値割り当てを真または偽への部分写像として定義しているが，ここでは 3 値形式で定義する．

充足されないという．節が**単位節** (unit clause) であるとは，節中の一つのリテラルの値が未割り当てで，残りのリテラルの値が 0 である場合をいう．しばしば，値割り当て ν を，変数 $x_i \in X$ とその値 $v_i \in \{0,1\}$ の組 (x_i, v_i) の集合として表す．値が未割り当ての変数は集合に含めないことで区別する．

SAT 問題を解く基本的な手続きである DPLL アルゴリズムを図 1 に示す．アルゴリズムは入力として CNF 形式の論理式 ψ と空の値割り当て ν を受け取り，1（充足可能）または 0（充足不能）を返す．まず関数 propagate を呼び出す．この関数は，値割り当て ν において単位節となる節があるならば（5 行目），その節を充足するために必然的に未割り当てのリテラルに真を割り当てる（6,7 行目）．この単位節による値割り当てを**単位伝播** (unit propagation) と呼ぶ．単位伝播により新たな単位節が生まれることがあるため，単位節がなくなるまでこの作業を繰り返す．その過程において，未割り当ての変数 x に対し，x を含む単位節と，$\neg x$ を含む別の単位節が同時に発生しうる．これを**矛盾** (conflict) と呼ぶ．x にいずれかの値を割り当てると ψ は充足されなくなるため関数を終了する（5 行目）．単位伝播の適用後，もし式 ψ が充足された（または矛盾が発生した）ならば 1（または 0）を返して終了する（2 行目）．それ以外の場合，まだ充足できていない節が存在するので，適当な変数選択ヒューリスティックスに基づいて未割り当ての変数 x を選択し（3 行目），x に真または偽を割り当てた場合を再帰的に試みる（4 行目）．この時点では，真と偽のいずれの値を割り当てるのが正しいのかは不明であるため，ここがバックトラックポイントになる．例えば，もし x に真を割り当て，その後の再帰的な呼び出しの中で矛盾が発生した場合は 0 を返すので，いずれは x に偽を割り当てることになる．従って単位伝播によって真偽値が確定しない変数に対しては真偽値割り当ての組合せを網羅的に試みることになる．その結果，もし充足可能な割り当てが見つからなければアルゴリズムは最終的に 0 を返す．

```
DPLL(ψ, ν)
  ψ: a CNF formula, ν: a variable assignment
  begin
1   ν = propagate(ψ, ν)
2   if ν(ψ) ≠ u then return ν(ψ)
3   choose an unassigned variable x
4   return DPLL(ψ, ν ∪ {(x,1)}) or DPLL(ψ, ν ∪ {(x,0)})
  end
propagate(ψ, ν)
  begin
5   while ∃C ∈ ψ(C is a unit clause) and ν(ψ) ≠ 0 do
6     l := the unassigned literal in C
7     ν := ν ∪ {(var(l), sign(l))}
8   return ν
  end
```

図 1 DPLL アルゴリズム

```
CDCL(ψ, ν)
  ψ: a CNF formula, ν: a variable assignment
  begin
1   dl := 0
2   loop
3     ν = propagate(ψ, ν)
4     if ν(ψ) = 1 then return 1
5     if ν(ψ) = 0 then
6       if dl = 0 then return 0
7       (C, bl) := analyze(ψ, ν)
8       ψ := ψ ∪ {C}
9       ν := {(x_i, v_i) ∈ ν | lv(x_i) ≤ bl}
10      dl := bl
11    else
12      dl := dl + 1
13      choose an unassigned variable x and its value v
14      ν := ν ∪ {(x, v)}
  end
```

図 2 一般的な CDCL アルゴリズム

3. 高速 SAT ソルバー

DPLL 手続きに基づく高速 SAT ソルバーの大きな特徴は，**矛盾からの節学習** (conflict-driven clause learning) であり，しばしば CDCL ソルバーと呼ばれる．アルゴリズムを示すにあたり用語を定義する．命題変数の値は，単位伝播によって割り当てられるか，変数選択ヒューリスティックスにより選択され割り当てられるかのいずれかである．前者を**含意** (implication) による割り当てと呼び，後者を**決定** (decision) による割り当てと呼ぶ．決定により値が割り当てられた変数を**決定変数** (decision variable)，決定変数の数を**決定レベル** (decision level) と呼ぶ．すべての変数は真偽値が割り当てられたときの決定レベルを保持するものとする．変数 x の値が割り当てられた決定レベルを $lv(x)$ と表記する．

図 2 に一般的な CDCL ソルバーのアルゴリズムを示す．まず決定レベルを表す変数 *dl* を 0 で初期化する．2〜14 行目はループであり，充足可能または不能が確定するまで繰り返す．まず単位伝播を適用する（3 行目）．もし式 ψ が充足されたならば 1 を返して終了する（4 行目）．逆に矛盾が発生した場合，もし決定レベルが 0 ならば明らかに充足不能であるので 0 を返して終了する（6 行目）．決定レベルが 1 以上の場合，矛盾が発生した原因を解析し（7 行目の関数 analyze），以後同様な矛盾の発生を防ぐため原因を否定した節 C を追加する（8 行目）．この追加される節を**学習節** (learnt clause) と呼ぶ．矛盾解析については 3·1 節で詳しく解説する．また原因の解析において，バックトラックすべき適切な決定レベル *bl* を同時に求めておき，その時点までバックトラックする（9 行目）．すなわち決定レベル *bl* より後の値割り当てを取り消す．単位伝播の適用後，ψ の値が u ならば，まだ充足できていない節が存在するので，決定レベルを 1 増加させ，適当な変数選択ヒューリスティックスに従い変数とその値を選択し，値を割り当てる（12〜14 行目）．

このアルゴリズムは DPLL アルゴリズムの一般形として捉えることができることができる．もし関数 analyze が (1) 矛盾の原因としてすべての決定変数とその値を選択し[*2]，(2) バックトラックすべきレベルとして “現在の決定レベル -1” を選択したとき，CDCL アルゴリズムは DPLL アルゴリズムと同じ動作をする．ただし，このようにして得られる学習節が単位伝播に使われることはその直後の 1 回しかなく，探索木における他の枝を刈る効果はない．従って SAT ソルバーの高速化のためには，より良い学習節を獲得することが必要不可欠である．次節でその方法を紹介する．

3·1 節学習と バックジャンプ法

SAT ソルバーにおける矛盾からの節学習アルゴリズムは，SAT ソルバー GRASP [Silva 99b] と Relsat [Bayardo 97] において提案され，Chaff [Moskewicz 01, Zhang 01] において現在広く使われている手法に洗練化された．ここでは例を用いてその手法を説明する．

〔例 1〕 以下の 12 個の節からなる CNF 式 ψ を考える．

$$
\begin{array}{ll}
C_1 = \{x_1, x_{13}\} & C_7 = \{\neg x_6, x_7, x_8\} \\
C_2 = \{\neg x_1, \neg x_2, x_{14}\} & C_8 = \{\neg x_4, \neg x_8, \neg x_9\} \\
C_3 = \{x_3, x_{15}\} & C_9 = \{\neg x_1, x_9, \neg x_{10}\} \\
C_4 = \{x_4, x_{16}\} & C_{10} = \{x_9, x_{11}, \neg x_{14}\} \\
C_5 = \{\neg x_5, \neg x_3, x_6\} & C_{11} = \{x_{10}, \neg x_{11}, x_{12}\} \\
C_6 = \{\neg x_5, \neg x_7\} & C_{12} = \{\neg x_2, \neg x_{11}, \neg x_{12}\}
\end{array}
$$

変数選択ヒューリスティックスは変数の添え字順に選択し，常に真を割り当てるものとする．単位節が存在しないため，まず x_1 を真に，次に x_2 を真とする．節 C_2 が単位節となり x_{14} が真となる．同様に x_3, x_4, x_5 に決定により順次真を割り当てる（決定レベルは 5 になる）．すると節 $C_5 \sim C_{11}$ にかけて単位伝播により次々と値が確定し，最終的に節 C_{12} が矛盾する（x_{12} が偽となるため）．

矛盾が発生すると CDCL アルゴリズムはその原因を解析する．まず単位伝播による含意関係を表す**含意グラフ** (implication graph) を作る（図 3）．含意グラフは非循環有向グラフである．頂点は変数への値割り当てを表す．例えば頂点 x_5 は命題変数 x_5 に真を割り当てたことを表し，$\neg x_7$ は x_7 に偽を割り当てたことを表す．またどの決定レベルにおいて値が割り当てられたのかを “@5” のように付記する．頂点に接続している辺は，その頂点が含意された理由を表す．例えば x_6 は，x_5 と x_3 が真になることで節 C_5 より含意される．白丸は現在の決定レベル（つまり 5）において値が割り当てられたことを示し，黒丸はそれ以前に割り当てられたことを示している．

次に含意グラフを 2 つの部分グラフに分割する．矛盾した変数を含む側を **conflict side**，他方を **reason side** と呼ぶ．ただし conflict side には黒丸を含めないものとする．

*2 例えば決定変数 x_1, x_2 の値がそれぞれ真と偽であったならば，矛盾の原因を $x_1 \wedge \neg x_2$ とし，学習節を $\neg x_1 \vee x_2$ とする．

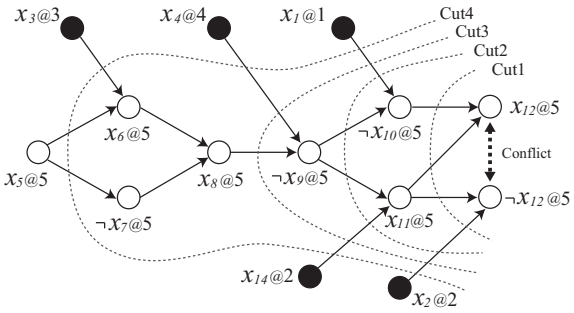


図 3 例 1 における含意グラフ

5

る．このような分割方法は幾通りも存在する．図 3 には 4 通りの分割を示している（Cut1〜4）．ある分割において，conflict side の頂点への辺を持っている reason side のすべての頂点は矛盾の原因を表す．例えば Cut3 の場合 $x_1, x_4, x_8, x_{14}, x_2$ の 5 つの頂点が conflict side への辺を持っており，これらがすべて真になると必ず $x_{12}, \neg x_{12}$ が含意されるので，$x_1 \wedge x_4 \wedge x_8 \wedge x_{14} \wedge x_2$ が矛盾の原因の 1 つである．矛盾の原因を否定すると節 $C' = \{\neg x_1, \neg x_4, \neg x_8, \neg x_{14}, \neg x_2\}$ が得られる．この節を CNF 式 ψ に追加しておけば，今後もし x_1, x_4, x_{14}, x_2 が真となったとき，C' は単位節となり x_8 に偽を割り当てることができる．つまり 図 3 における頂点 x_8 以降の矛盾に至る探索を避けることが可能となる．このようにして得られる節を**学習節** (learnt clause) と呼ぶ．学習節は ψ の論理的帰結である．実際 C' は，節 $C_{12}, C_{11}, \ldots, C_8$ を順に融合することで得られる．従って ψ に C' を追加してもその充足可能性は変わらない（健全性）．また矛盾が起きるたびに学習節を獲得し追加するため，上で示したように探索空間は徐々に狭まることになり，最終的には充足可能または不能を示すことができる（完全性）．

含意グラフの分割方法は幾通りもあるので，1 つの矛盾から複数の学習節を得ることができるが，矛盾のたびにすべての学習節を保持することは現実的ではない．Zhang らは SAT ソルバーにおける様々な学習スキーマを評価し，以下で述べる first UIP により分割することで効果的な学習節が獲得できることを示した [Zhang 01].

まず UIP を定義する．含意グラフにおける **UIP** (unique implication point) とは，現在の決定レベルにおける決定変数を表す頂点から矛盾した頂点へ至るパスを考えたときに，すべてのパスが必ず通る頂点のことである．図 3 では，UIP は 3 つある（$x_5, x_8, \neg x_9$）．決定変数は必ず UIP となるので，少なくとも 1 つは UIP が存在する．ところで矛盾を解消するためには，少なくとも 1 つ前の決定レベルにバックトラックする必要がある．このとき含意グラフを UIP の直後で分割することにより得られる学習節は，必ず単位節になる（UIP 以外のリテラルはすでに偽であるため．例えば C' では，x_8 のみが未割り当てで他は偽になる）．逆に UIP でない頂点の直後で分割し

た場合，学習節は単位節とならない（Cut1）．変数の値を確定できる機会があるのであれば，それを利用しない手はないので UIP の直後で分割するのがよい．**First UIP** とは最も矛盾に近い UIP のことである（図 3 では $\neg x_9$）．First UIP の直後で分割することにより得られる学習節の長さは，他の UIP より得られる学習節よりも短いか等しい（矛盾導出のために必要となる黒丸の数を減らせるため）．節は短ければ短いほど値割り当てに対する強い制約となり，探索空間を狭めることができる．Zhang らは，first UIP より得られる学習節のみを追加することが最も効果的であることを実証している [Zhang 01]．例 1 では first UIP による分割は Cut2 であり，追加される学習節は $C'' = \{\neg x_1, x_9, \neg x_{14}, \neg x_2\}$ となる．さらに MiniSat では学習節の簡単化も行う [Sörensson 09]．図 3 では省略したが，頂点 x_1 と x_2 は頂点 x_{14} への辺を持つ（節 C_2 参照）．従って x_1 と x_2 があれば x_{14} は不要である．よって C'' は $C''' = \{\neg x_1, x_9, \neg x_2\}$ と簡単化できる（これは C'' と C_2 を融合することに等しい）．

次にバックトラックすべきレベルを決定する．DPLL 手続きでは，もし決定レベル l において矛盾が発生した場合，レベル l の決定変数の値について真と偽の両方試していないのであれば残りの値を試し，それでも矛盾が発生した場合にレベル $l-1$ にバックトラックする．レベル $l-1$ でも同様に，もし決定変数の値を両方試していないのであれば他方の値を試し，すでに両方試していればさらに $l-2$ へバックトラックする．このように 1 レベルずつやり直す手法を **時間順バックトラック法** (chronological backtracking) という．この手法の問題点は，もし矛盾の原因が浅い決定レベルにあった場合，そこに戻るまでに無駄な探索を指数関数的に繰り返す点にある．

一方 CDCL ソルバーでは，複数レベルを一度に戻る **非時間順バックトラック法** (non-chronological backtracking) または **バックジャンプ法** (backjumping) と呼ばれる手法をとる．先の学習節 C'' を見ると，決定レベル 1,2,5 において値が割り当てられたリテラルしか存在しないことが分かる．学習節は論理的帰結であるので，リテラル x_9 の値は，本来ならばレベル 2 の時点で確定できたことを意味する．そこで Chaff や MiniSat では，レベル 3,4,5 をやり直すことなく直接レベル 2 までバックトラックし，その後単位伝播によって x_9 に真を割り当てる[*3]．

学習節は，探索しても無駄な空間を取り除くための有用な知識ではあるが，矛盾のたびに学習するため節数の増大をもたらす．問題によっては元の節数を超えることがしばしばある．節数の増大は，メモリ空間の圧迫や単位節検出のためのコストの増加をもたらすため，役に立っていない学習節は削除することが望ましい．例えば MiniSat では，学習節に **活性度** (activity) という指標を持たせ，学習節が生成されたとき，または矛盾の原因として使用されるたびにそれを向上させている．また向上の度合いを徐々に増加させることで新しい学習節ほど活性度が高くなるようにしている．そして学習節の数がある閾値を超えたとき，活性度の低い節を削除する．この閾値もまた徐々に増加されるため，完全性が失われることはない．

3·2 監視リテラル

SAT ソルバーの実行時間の 70〜90% は単位伝播の処理で占められるため [Moskewicz 01]，効率の良い単位節の検出アルゴリズムは高速化のための重要な要素である．

単位節を検出する素朴な方法として，各変数 x_i に対し，x_i を含む節のリストを用意しておき，x_i に値が割り当てられたときそのリストを走査することで状態の変化した節のみを確認する（つまり単位節かどうか，充足されているか，矛盾しているか，そのいずれでもないかを確認する）手法が考えられる．この方法は GRASP や Relsat などのソルバーで用いられているが，節リストの走査の目的が単位節または矛盾した節の検出にあるにもかかわらず，ほとんどの節はそのいずれでもなく無駄が多いという問題点がある．充足済みの節をリストから取り除くことで走査すべき節数を削減することができるが，この場合，バックトラック時に節リストを復元する必要がありやはり手間がかかる．

Chaff で導入された **監視リテラル** (watched literals) による単位節の検出方法 [Moskewicz 01] は，上記の問題点を解決する簡潔で優れた手法である．節 $C = \{l_1, \ldots, l_n\}$ とする．C が単位節になる直前の状態は，C 中のリテラルのうち 2 つのみが未割り当てで（これを l_i, l_j とする），残りが偽の場合である．l_i または l_j のいずれか（または両方）に偽が割り当てられたとき C は単位節となる（または矛盾する）．従って単位節または矛盾の検出のために C 中のすべてのリテラルを常に調べる必要はなく，未割り当てのリテラル 2 つのみを監視すれば十分である．

図 4 に監視リテラルの動作例を示す．各節は監視中のリテラルを指し示す 2 つのポインター w_1, w_2 を持つ．w_1, w_2 は異なるリテラルを監視するものとする．監視中のリテラルのいずれかが真ならば，その節はすでに充足されているので何もする必要はない．(a) 最初，先頭の 2 つのリテラル x_1 と $\neg x_2$ を監視しているものとする．(b) 決定レベル 1 において w_1 が監視中のリテラル x_1 が偽になると，w_1 は偽でないリテラルを探し（$\neg x_3$），そこに移動する．(c) 同様に決定レベル 2 において w_2 が監視中のリテラル $\neg x_2$ が偽になると，w_2 は偽でないリテラル（x_4）に移動する．(d) 監視していないリテラル $\neg x_5$ の値が変化しても何もする必要はない．(e) 決定レベル 4 において w_1 が監視中のリテラル x_3 が偽になっ

*3 この例では，決定レベル 2 まで戻らなくとも，レベル 5 をやり直すことで（すなわち x_5 に偽を割り当てることで）充足可能な真偽値割り当てを発見することができる．すなわち Chaff や MiniSat では，探索が完全に行き詰まった（そのレベルで必ず矛盾する）と判断できなくても積極的にバックトラックするといえる．GRASP では，探索が完全に行き詰まったと判断してから原因を解消できるレベルまでバックトラックする．

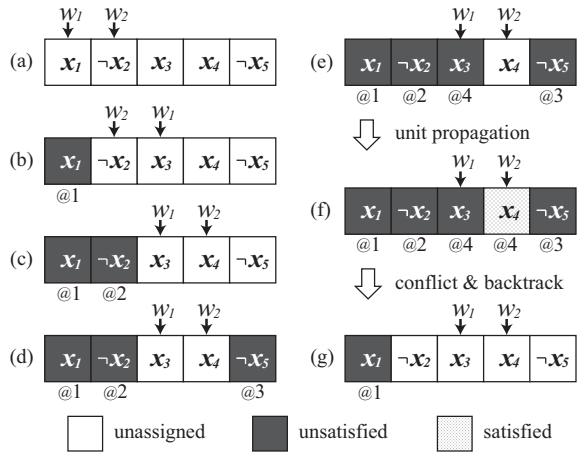


図 4 監視リテラルによる単位節の検出とバックトラックの様子

た場合，w_1 は偽ではないリテラルを探すが存在しないため，この節が単位節になったと判断できる．(f) そこで単位伝播により x_4 を真とする．(g) その後どこかの決定レベルにおいて矛盾が発生し，レベル 4 よりも前のレベルまでバックトラックすることになったとする（図ではレベル 1，つまり (b) に戻っている）．この場合も w_1, w_2 を動かす必要はない．なぜならレベル 4 より前のレベルでは，x_3, x_4 は必ず未割り当てになるからである．また，レベル 4 以降に戻る場合は当然何もする必要はない．

監視リテラル手法の利点として，(1) ある変数 x_i に値が割り当てられたとき，x_i を含むすべての節を走査するのではなく，x_i を監視中の節のみを走査するだけで済むため，調べる節数を大幅に抑えることができること，(2) バックトラック時には単に変数を未割り当てに戻すだけで良いことが挙げられる．さらに (3) CPU データキャッシュのミスヒット率を大幅に低減する効果もある [Moskewicz 01, Zhang 03]．変数 x_i に値を割り当てることは x_i を監視中の節数が大きく減ることを意味する．矛盾によるバックトラックが起きると，最近値を割り当てた変数に再び値を割り当てることがしばしばある．このとき x_i を監視中の節数は少なくなっているため，最初に値を割り当てたときよりも高速に値を割り当てることが可能になる．これは節を格納しているメモリへのアクセス回数を大きく減らし，その結果キャッシュのミスヒット率が低減する．Zhang らは，GRASP などの従来手法と比較して，ミスヒット率の低減だけでも約 3 倍の速度の向上が得られることを示している [Zhang 03].

3·3 変数選択ヒューリスティックス

変数選択ヒューリスティックスは，現在の値割り当てにおいて単位節が存在しない場合に，どの変数にどの値を割り当てるかを決めるために利用される．変数選択ヒューリスティックスは SAT ソルバーの性能に大きな影響を与えるため，これまでに多くの手法が提案されているが，その基本的な方針は頻出するリテラルに真を割り当てることにある．よく知られたヒューリスティックスである MOMS (Maximum Occurrence in clauses of Minimum Size) [Freeman 95] や BOHM [Böhm 96] では，短い未充足の節に多く出現するリテラルを優先して選択する．GRASP では，節の長さにかかわらず，未充足の節に最も多く出現するリテラルを選択し，それを真としている [Silva 99b]．初期のヒューリスティックスの性能比較は文献 [Silva 99a] に良くまとめられている．

これらのヒューリスティックスでは，各リテラル l に対し，未充足の節における l の出現回数を求める必要があるが，これはコストのかかる計算であり，また前節の監視リテラルとの相性も悪い．例えばリテラルの出現頻度表を用意したとしても，節が充足されるたびに，またはバックトラックが起きるたびにその表を更新する必要がある．また監視リテラルでは，ある変数 x_i に対して，x_i を含むすべての節のリストではなく，x_i を監視中の節のリストしか保持しないため，x_i に値が割り当てられたときに充足されたすべての節を知ることは難しい．

そこで監視リテラルを導入した SAT ソルバー Chaff では，VSIDS (variable state independent decaying sum) と呼ばれるヒューリスティックスを採用している [Moskewicz 01]．(1) まず各リテラルごとにその優先度を表すカウンタを用意し，0 で初期化する．(2) 学習節が追加されるたび，その節に含まれるリテラルのカウンタを増加させる．(3) 変数選択においては，最も高い優先度を持つ未割り当てのリテラルを選択し，それを真とする．また，一定の確率で優先度にかかわらずリテラルをランダムに選択する．そして (4) 定期的にすべてのカウンタの値を定数で割り減少させる．VSIDS では，未充足の節中におけるリテラルの出現回数に依存することなく，最も高い優先度を持つリテラルが選択される（もしステップ (4) がなければ，学習節が追加されるたびに，それに含まれるリテラルの優先度が増加するので，優先度はリテラルの出現回数の大まかな近似にはなっている）．(2) および (4) より，VSIDS では最近追加された学習節に含まれるリテラルが優先的に選択されるため，学習節を充足するようにリテラルを選択する働きがある．これは局所的な探索を促し，その領域における矛盾解消が促進されることになる．矛盾が発生しない限りリテラルの優先順位に変化はないため，VSIDS は，MOMS や BOHM のように多くの短い節を充足することよりも，矛盾解消を目的としたヒューリスティックスであるといえる．また VSIDS の計算コストは非常に少ない．VSIDS では学習節の追加時に，その節中のリテラルの優先度を増加させるだけでよいからである[*4]．Moskewicz らは，Chaff が VSIDS を採用することにより，ほとんどの問題において劇的に性能の改善が得られたと述べている [Moskewicz 01].

VSIDS とその派生版は，Chaff だけでなく，MiniSat

*4 変数選択時の手間は，最も評価の高いリテラルを選択する必要があるため，他の手法でも同様である．

や BerkMin [Goldberg 07]，RSat [Pipatsrisawat 07] など多くの CDCL ソルバーにおいて採用されている．例えば MiniSat では，リテラル l と $\neg l$ を区別せずに変数に優先度を持たせている点が Chaff と異なる．選択された変数には常に偽が割り当てられる．また最も優先度が高い変数を効率良く求めるため，すべての変数を優先度に基づいてヒープ木に格納しておき，優先度が更新されたときに木のノードを適切な位置に移動させることで，変数全体を走査することを避けている．

3·4 リスタート

SAT 問題の変数番号を付け替えるだけで，それまで数秒で解けていた問題が非常に困難になったり，またはその逆になることがしばしばある．また変数選択ヒューリスティックスにランダムな要素を含めた場合にも，実行によって一瞬で解けたり，何時間待っても解けない場合がある．このように，SAT ソルバーの実行時間が無視できない確率で非常に長くなることは，裾の重い (heavy-tailed) 振る舞いとして知られている [Gomes 97]．この振る舞いによる不安定さを除去するために，Gomes らは，探索を比較的短い間隔で最初からやり直すリスタート戦略が極めて有効であることを示している [Gomes 98]．CDCL ソルバーでは，矛盾の発生回数がある閾値を超えると決定レベル 0 までバックトラックし，探索をやり直す．このとき学習節は破棄しないため，同様の探索失敗を繰り返すことはない．また変数の優先度カウンタも再初期化しないため，リスタート前と同じ順序で変数を選択することもない．またアルゴリズムの完全性を維持するため，リスタートの間隔を徐々に延ばすようにする．

事前に実行時間の分布が分かっているならば，実行時間の期待値を最小化するたリスタートの最適な間隔が存在することが知られている [Luby 93]．Kautz らは，実行時のソルバーの振る舞いから得られる情報に基づいて実行時間の分布を予測し，最適なリスタート戦略を動的に導く手法を示している [Kautz 02]．

3·5 最新の動向

2009 年の SAT 競技会における Application 部門の UNSAT と SAT+UNSAT 部門でそれぞれ 1 位と 2 位を獲得した SAT ソルバー glucose について簡単に紹介する．

3·1 節で述べたように学習節の過度の増加は，探索効率の悪化をまねくため，良い学習節のみを節データベースに残していくことが望ましい．Zhang らは，矛盾がより浅い決定レベルで発見されることを促進する学習節が良い学習節であると述べており [Zhang 01]，Audemard らは，そのような節を効果的に判断する尺度として **LBD** (literal block distance) を提案した [Audemard 09]．ある学習節の LBD は，その節内のブロック数で与えられる．ここでブロックとは，同じレベルで値が割り当てられた変数の集合をいい，構造的な SAT 問題では，意味的に深く関連する変数から構成される傾向がある．報告では first UIP による学習節の中から LBD が少ないものを積極的に活用することで，以降の探索における矛盾をより浅い決定レベルで発見することが期待できると述べられている．特に 2 種類のブロックをつなぐ働きを促す LBD=2 の学習節は glue clauses と呼ばれ重要としている．

このアイデアを基に glucose [Audemard 09] は，LBD を用いた学習節データベース管理方法を実装している．従来手法と比較して学習の効果を下げることなく保持する学習節を大幅に削減することに成功しており，その結果 SAT 競技会における前述の成績を収めている．

4. MiniSat の利用方法

現在幅広く使用されている MiniSat の開発の動機は興味深い．通常研究者がプログラムを開発する目的は，新しく自らが提案する手法の性能を実証することが一般的である．しかし Eén と Sörensson は，MiniSat の開発において，既存のアルゴリズムをいかに簡潔に効率良く実装するかを主眼とし，小さな SAT ソルバーを開発することで，拡張容易な分かりやすいソルバーを SAT コミュニティに提供することを目的としていた [Eén 03a]．MiniSat の最初のバージョンは，C++ で 600 行程度（コメントを除く）であり，最新の MiniSat2 でも 1500 行程度である．ソースコードは読みやすく，拡張しやすい設計になっている．

MiniSat は MIT ライセンスのもとで配布されており[*5]，C++ コンパイラさえあれば Linux, Mac, Windows など多くの環境で実行可能である．実行方法を以下に示す．

```
$ minisat [option] <infile> <outfile>
```

入力ファイルは省略可能で，その場合は標準入力から SAT 問題を読み込む．SAT 問題は DIMACS CNF 形式[*6] で記述する．例えば CNF 式 $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$ は次のように与える：

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

1 行目 “p cnf” の後に続く数字は変数の数と節数を表し，2 行目以降は節を表す．“-” は否定，1 以上の整数は変数，0 は各節の終わりを示している．SAT 問題のサイズはしばしば巨大になるので MiniSat では gzip 形式で圧縮されたファイルも入力可能である．実行後は，充足可能または不能を出力する．出力ファイルが指定されている場合，充足可能であれば真偽値割り当てが出力される．オプションでは，変数選択時に割り当てる真偽値やランダムに変数を選択する確率などのパラメータを指定できる．

*5 http://minisat.se/
*6 http://www.satlib.org/Benchmarks/SAT/satformat.ps

```
   #include<stdio.h>
   #include<Solver.h>
   int main() {
1    Solver S;
2    vec<Lit> c1, c2, c3, c4;

3    S.newVar(); S.newVar(); S.newVar();
4    c1.push(Lit(0)); c1.push(~Lit(1));
5    S.addClause(c1);
6    c2.push(Lit(1)); c2.push(Lit(2));
7    S.addClause(c2);
          ⋮
8    if (S.solve()) {
9      printf("SAT\n");
10     for (int i=0; i < S.nVars(); i++) {
11       if (S.model[i]==l_False)
12         printf("-");
13       printf("%d ", i+1);
14     }
15     printf("\n");
16   } else
17     printf("UNSAT\n");
18   return 0;
    }
```

図 5 MiniSat ライブラリの利用例

MiniSat はライブラリとしても利用できるように設計されており，他のプログラムから簡単に呼び出すことができる．その場合，ヘッダファイル Solver.h をインクルードし，“make lib” により生成されるライブラリ libminisat.a をリンクすればよい．図 5 は上で示した SAT 問題を解くコード例である．1, 2 行目では，Solver オブジェクトと節を格納するためのオブジェクトを生成している．3 行目では，SAT 問題が 3 変数からなることを宣言している．4〜7 行目で節を登録している（一部省略）．DIMACS CNF 形式では変数番号は 1 から始まるが，MiniSat 内部では変数番号は 0 から始まることに注意されたい．メソッド solve を呼び出すと MiniSat は SAT 問題を解く（8 行目）．充足可能の場合は真が返るので，このプログラムでは標準出力に “SAT” とその変数割り当てを表示する．充足不能の場合は “UNSAT” を表示する．

SAT ソルバーの利用にあたっては，しばしば複数の SAT 問題を連続して解く必要に迫られる場合がある．SAT は判定問題であるため，プランニング問題や制約最適化問題などを SAT に変換して解く場合，プラン長を最適化したり目的関数を最適化するために，複数の SAT 問題を解くことになるためである．このような場合，学習節を再利用することで，より効率良く解くことが可能である [Eén 03b]．SAT 問題の系列を $P_1, P_2, P_3, \ldots$ とする．このとき単位節を除いて $P_i \subseteq P_j\ (i < j)$ となるならば，P_i を解くときに生成された学習節を P_j を解くために利用できる [Nabeshima 06]．すなわち，単位節が除去されることがあったとしても，長さ 2 以上の節が単調に増加する場合，学習節を再利用できる．P_j を解く際に P_i の学習節を再利用することで，探索しても無駄であることが分かっている空間を再度探索し直すことを避けることが可能になる．MiniSat では，除去される可能性のある単位節の集合を solve メソッドの第 1 引数に指定する．Solver オブジェクトは solve の呼び出し後も学習節を保持しているため，長さ 2 以上の節を登録後，再度 solve を呼び出せば，それまでに学習した学習節を再利用できる．

5. 先読み型 SAT ソルバー

ランダムに生成された問題（特に充足不能な問題）に対しては，CDCL ソルバーよりも，Satz [Li 97], k-cnfs [Dequen 06] そして March [Heule 08] に代表されるような **先読み型 SAT ソルバー** (look-ahead based SAT solver) が適していることが報告されている．

先読み型ソルバーも，CDCL ソルバーと同じく DPLL 手続きを基にした系統的 SAT ソルバーであるが，その探索方法は次のように異なっている．CDCL ソルバーは，VSIDS 等の計算コストの小さい方法で変数選択を行い，学習節を用いた矛盾の解消に重きをおく．しかし幅広く実装されている first UIP による節学習は，構造的な問題に対しては矛盾が発生する決定レベルを浅くする効果があるにもかかわらず，ランダムな問題に対しては同様の効果が得られないことが示されている [Audemard 09]. それに対して先読み型ソルバーは，計算コストが大きくても，変数選択の結果生じる単位伝播によってより多くの単位節を作ることができる変数を選択することに重きをおく．これによりランダムな問題に対しても探索空間を小さく保ちながら充足可能性の判定を行うことができる．

次に先読み型ソルバーにおいて幅広く利用されている**単位伝播ヒューリスティックス** (unit propagation heuristics) を用いた探索手続きを説明する．まず各変数に対し，真を割り当てた場合と偽を割り当てた場合をそれぞれ試す．このとき，(a) 単位伝播により値が確定した変数の数と，(b) 偽を割り当てられたリテラル数が増加した（すなわち短くなった）未充足の節数を測定し，評価値を算出する．(a) (b) ともに数が多い方が高い評価値を得る．もしこの先読み手続きにおいて，ある変数に真（または偽）を割り当てたことにより矛盾が発生した場合には，その変数には偽（または真）が必然的に割り当てられ，再度単位伝播を行う．この場合，単位伝播によって値を割り当てられる変数が増加することがあるので，その時には評価値の再計算のために先読み手続きを再度実行する．これを評価値の更新がなくなるまで繰り返す．その後，真と偽両方の評価値を組み合わせた上で変数が選択される．

以上が基本的な動作であるが，すべての変数を先読みの対象とすると計算コストが非常に高くつく．そこで対象とする変数を絞り込む手法として，Satz では 長さ 2 の節に多く含まれている変数を優先して対象変数とする Prop_z [Li 97] と呼ばれる手法が実装されている．また March では CRA (clause reduction approximation) [Heule 08] と呼ばれる変数割り当てによって長さが短くなる節

数の近似的指標を計算することで変数の優先度を決定し，対象変数を絞り込んでいる．

先読み型ソルバーも CDCL ソルバーと同様に高速化技術が研究されており，近年ではランダムな問題だけでなく構造的な問題に対しても効率よく探索できることが報告されるようになった [Kullmann 02, Heule 08]. 特に March は 2004 年以降の SAT 競技会 において Random 部門だけでなく Crafted 部門でも複数回入賞する成績を収めており，このことは先読み型ソルバーが構造的な問題に対しても有効であることを裏付けている．

6. お わ り に

本稿では，高速 SAT ソルバーにおける代表的な要素技術について説明を行った．近年 SAT ソルバーは，[井上 10] で示されているように，幅広い分野において使用され成果をあげている．このことは他分野のソルバーと比較しても SAT ソルバーが高速である事を裏付けている．その理由の 1 つに SAT 問題の形式の簡潔さが挙げられる．この簡潔さが種々の高速化手法を生む土台となっており，またコンパクトな実装を可能にしている．監視リテラル等の技術により CPU キャッシュのミスヒットを抑えていることも，現在の計算機システムに適合した高速化の要因の 1 つである．そして SAT の簡潔さは，多くの研究者を引きつける魅力にもなっている．今後のさらなる高速化や大規模化に向けて，本稿がその一助になれば幸いである．

◇ 参 考 文 献 ◇

[Audemard 09] Audemard, G. and Simon, L.: Predicting Learnt Clauses Quality in Modern SAT Solvers, in *Proceedings of IJCAI-2009*, pp. 399–404 (2009)

[Bayardo 97] Bayardo, R. J. and Schrag, R. C.: Using CSP Look-Back Techniques to Solve Real-World SAT Instances, in *Proceedings of AAAI-97*, pp. 203–208 (1997)

[Böhm 96] Böhm, M. and Speckenmeyer, E.: A Fast Parallel SAT-Solver - Efficient Workload Balancing, *Annals of Mathematics and Artificial Intelligence*, Vol. 17, No. 3-4, pp. 381–400 (1996)

[Davis 62] Davis, M., Logemann, G., and Loveland, D.: A Machine Program for Theorem Proving, *Communications of the ACM*, Vol. 5, No. 7, pp. 394–397 (1962)

[Dequen 06] Dequen, G. and Dubois, O.: An Efficient Approach to Solving Random k-sat Problems, *Journal of Automated Reasoning*, Vol. 37, No. 4, pp. 261–276 (2006)

[Eén 03a] Eén, N. and Sörensson, N.: An Extensible SAT-solver, in *Proceedings of SAT-2003*, pp. 502–518 (2003)

[Eén 03b] Eén, N. and Sörensson, N.: Temporal Induction by Incremental SAT Solving, *Electronic Notes in Theoretical Computer Science*, Vol. 89, No. 4, pp. 543–560 (2003)

[Freeman 95] Freeman, J. W.: *Improvements to Propositional Satisfiability Search Algorithms*, PhD thesis, University of Pennsylvania, Philadelphia, PA, USA (1995)

[Goldberg 07] Goldberg, E. and Novikov, Y.: BerkMin: A Fast and Robust SAT-Solver, *Discrete Applied Mathematics*, Vol. 155, No. 12, pp. 1549–1561 (2007)

[Gomes 97] Gomes, C. P., Selman, B., and Crato, N.: Heavy-Tailed Distributions in Combinatorial Search, in *Proceedings of CP-97*, pp. 121–135 (1997)

[Gomes 98] Gomes, C. P., Selman, B., and Kautz, H. A.: Boosting Combinatorial Search Through Randomization, in *Proceedings of AAAI-98*, pp. 431–437 (1998)

[Heule 08] Heule, M.: *SmArT solving: Tools and Techniques for Satisfiability Solvers*, PhD thesis, Delft University of Technology, Delft, The Netherlands (2008)

[井上 10] 井上 克巳, 田村 直之：SAT ソルバーの基礎, 人工知能学会誌 (2010)

[Kautz 02] Kautz, H. A., Horvitz, E., Ruan, Y., Gomes, C. P., and Selman, B.: Dynamic Restart Policies, in *Processing of AAAI-02*, pp. 674–681 (2002)

[Kullmann 02] Kullmann, O.: Investigating the Behaviour of a SAT Solver on Random Formulas, Tech. report csr 23-2002, computer science report series, University of Wales Swansea (2002)

[Li 97] Li, C. M. and Anbulagan, : Look-Ahead Versus Look-Back for Satisfiability Problems, in *Proceedings of CP-97*, pp. 341–355 (1997)

[Luby 93] Luby, M., Sinclair, A., and Zuckerman, D.: Optimal Speedup of Las Vegas Algorithms, *Information Processing Letters*, Vol. 47, No. 4, pp. 173–180 (1993)

[Moskewicz 01] Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., and Malik, S.: Chaff: Engineering an Efficient SAT Solver, in *Proceedings of DAC-01*, pp. 530–535 (2001)

[Nabeshima 06] Nabeshima, H., Soh, T., Inoue, K., and Iwanuma, K.: Lemma Reusing for SAT based Planning and Scheduling, in *Proceedings of the International Conference on Automated Planning and Scheduling 2006 (ICAPS'06)*, pp. 103–112 (2006)

[Pipatsrisawat 07] Pipatsrisawat, K. and Darwiche, A.: RSat 2.0: SAT Solver Description, Technical Report D–153, Automated Reasoning Group, Computer Science Department, UCLA (2007)

[Silva 99a] Silva, J. P. M.: The Impact of Branching Heuristics in Propositional Satisfiability Algorithms, in *Proceedings of the 9th Portuguese Conference on Artificial Intelligence*, pp. 62–74 (1999)

[Silva 99b] Silva, J. P. M. and Sakallah, K. A.: GRASP – A Search Algorithm for Propositional Satisfiability, *IEEE Transactions on Computers*, Vol. 48, pp. 506–521 (1999)

[Sörensson 09] Sörensson, N. and Biere, A.: Minimizing Learned Clauses, in *Proceedings of SAT-2009*, pp. 237–243 (2009)

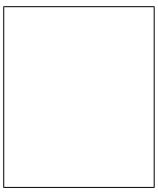
[Zhang 01] Zhang, L., Madigan, C. F., Moskewicz, M. W., and Malik, S.: Efficient Conflict Driven Learning in Boolean Satisfiability Solver, in *Proceedings of ICCAD-01*, pp. 279–285 (2001)

[Zhang 03] Zhang, L. and Malik, S.: Cache Performance of SAT Solvers: a Case Study for Efficient Implementation of Algorithms, in *Proceedings of SAT-2003*, pp. 287–298 (2003)

〔担当委員：××○○〕

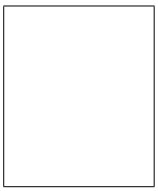
19YY 年 MM 月 DD 日　受理

著 者 紹 介



鍋島　英知（正会員）

2001 年神戸大学大学院・自然科学研究科情報メディア科学専攻博士課程修了．同年山梨大学工学部コンピュータ・メディア工学科助手．現在，同大学大学院・医学工学総合研究部准教授．博士 (工学)．アクション理論，プランニング，定理自動証明プログラム，WEB 知的処理などの研究に従事．2004 年 FIT 優秀論文賞受賞．



宋　　剛秀（学生会員）

2004 年神戸大学工学部電気電子工学科卒業．2006 年同大学院自然科学研究科電気電子工学専攻修了．サントリー（株）を経て，2008 年より総合研究大学院大学複合科学研究科情報学専攻博士後期課程に在学．SAT 問題を含む制約充足問題の研究に従事．2009 年度人工知能学会全国大会優秀賞受賞．