1D5-OS-02b-2

CDCL ソルバーにおける ZDD を利用した節圧縮表現の導入

Towards for the introduction of ZDD for clause database in CDCL solvers

後藤 優也 *1　　鍋島 英知 *2
Yuuya Gotou　　Hidetomo Nabeshima

*1山梨大学工学部コンピュータ理工学科
Department of Computer Science and Engineering, Faculty of Enginnering, University of Yamanashi

*2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

In this paper, we consider a CDCL solver which uses ZDDs as a clause database. The purpose is to handle a large number of clauses and realize an efficient unit propagation on them. Aloul et al. have proposed a ZDD based DPLL solver, which constructs ZDD from a given CNF, and then converts it to DAG, and realizes propagation mechanism on DAG. To extend it to CDCL, we show some issues and approach to solve them. We introduce a prototype of ZDD based CDCL solver and show the preliminary experimental results.

1. はじめに

SAT 問題とは命題論理式の充足可能性判定問題であり，ハードウェア・ソフトウェア検証やプランニング，スケジューリングなど様々な分野で応用されている．SAT 問題を解く SAT ソルバーの性能は，近年様々な高速化技術により飛躍的に向上している．多くの SAT ソルバーでは，単位伝播と呼ばれる単位節を充足するための必然的な割り当て処理が行われており，この単位伝播処理はソルバーの実行時間の 70〜90 %を占めているため [5]，単位伝播の性能向上はソルバーの性能向上に繋がる．また，ZDD[1] と呼ばれる組合せ集合を効率よく圧縮表現するデータ構造がある．この ZDD を DPLL ソルバーと組合せ，性能向上を示した研究 [6] がある．そこで本稿ではこの研究を近年の主流ソルバーである CDCL ソルバーで実装・評価を行う．

本稿の構成は以下の通りである．まず 2 章で SAT 問題と SAT ソルバーについて説明する．3 章では ZDD について説明する．4 章では既存手法について説明し，5 章において拡張にあたっての問題点を挙げ，6 章において提案手法を紹介．7 章では提案手法の性能評価と考察を行なう．8 章でまとめと今後の課題について述べる．

2. SAT 問題と SAT ソルバー

2.1 SAT 問題

SAT 問題とは，ある命題論理式を真にする変数割り当ての存在を判定する問題である．SAT 問題は通常 **CNF 式** (Conjunctive Normal Form) で与えられる．命題変数及びその否定を**リテラル**，リテラルの選言を**節**，節の連言を **CNF 式**と呼ぶ．以下にその例を示す．

$$(a \vee b \vee c) \wedge (\neg a \vee \neg c \vee d) \wedge (b \vee \neg d \vee e)$$

節 C が，部分的な値割り当てにおいて**単位節**であるとは，C 中の 1 つのリテラルのみが未割り当てで，その他のリテラルが偽であるときをいう．

連絡先: 後藤　優也，山梨大学工学部コンピュータ理工学科，〒 400-8511 山梨県甲府市武田 4-3-11，E-mail : yuuya@nabelab.org

SAT 問題を解く SAT ソルバーは，与えられた CNF 式を充足する変数の真偽値割り当てが存在するならば，**充足可能** (SAT) とその真偽値割り当てを出力し，存在しないならば**充足不能** (UNSAT) を出力する．

2.2 DPLL

DPLL[4] は，SAT ソルバーの基本的なアルゴリズムであり，バックトラックを用いて探索を行う．DPLL では，まず単位節を探す．単位節が存在した場合，その節を充足させるために未割り当てのリテラルに真を割り当てる．この単位節による変数の値割り当てを**単位伝播** (unit propagation) と呼ぶ．これを単位節がなくなるまで繰り返し行なう．単位節が存在しない場合，適当な変数選択ヒューリスティックスにより変数 x を選択し，x に真または偽を割り当てる．DPLL はこの単位伝播と変数選択を繰り返し動作する．単位伝播処理の途中で，単位節 x と $\neg x$ が同時に発生することがある．これを**矛盾** (conflict) と呼び，矛盾が発生したら直前の変数選択による値割り当てをやり直すためバックトラックする．逆に矛盾することなくすべての節を充足できたならば SAT を返し，網羅的に探索しても充足可能な割り当てが見つからなかった場合は UNSAT を返す．

2.3 CDCL

最新の SAT ソルバーは DPLL を拡張した CDCL(Conflict-Driven Clause Learning) アルゴリズム [3] に基づく．DPLL で探索を行なうと，探索中矛盾が多く発生する．矛盾が発生した時に，その矛盾の原因を解析し，同じ矛盾の発生を防ぐための節を学習する．この節を**学習節**と呼ぶ．学習節を元問題に追加することで，過去に矛盾の原因となった変数割り当てを起こさないため，同じ矛盾の発生を防ぐことができ，探索空間を狭めることができる．

2.4 監視リテラル

単位伝播処理はソルバーの実行時間の 70〜90 %を占めているため [5]，単位伝播の性能向上はソルバーの性能向上に繋がる．単位伝播処理に必要な単位節を効率よく発見する手法として監視リテラル [5] がある．単位伝播処理を考えたとき，ある節が単位節となる直前の状態というのは，節内のリテラルの 2 つが未割り当てであり，他すべてに偽が割り当てられているときである．この 2 つの未割り当てのリテラルのいずれかに偽

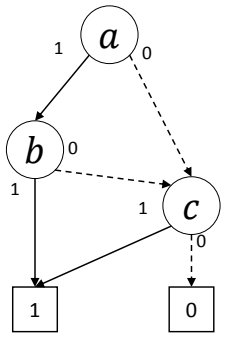

図 1: 集合 $S = \{\{a, b\}, \{a, c\}, \{c\}\}$ を表す ZDD

が割り当てられたときに，この節は単位節となる．

つまり単位節検出には節内のすべてのリテラルを調べる必要はなく，未割り当ての 2 つのリテラルのみを監視していればよい．監視していないリテラルに値割り当てが行われてもその節が単位節になることはないので，その節を調べる必要はない．監視しているリテラルに偽が割り当てられた場合のみ，新たに監視対象として未監視のリテラルで真または未割り当てのリテラル l を探す．もし l が存在すればそれを監視し，l が存在しなければこの節は単位節であると検出できる．

3. ZDD

ZDD(Zero-suppressed binary Decition Diagram)[1][2] とは，組合せ集合を効率よく表現するデータ構造で，特に疎な組合せ集合には非常に圧縮効果がある．

ZDD は有向非巡回グラフ (DAG: Directed Acyclic Graph) で表され，各接点はアイテム名を表す変数名と，アイテムを選ぶことを意味する 1-枝と，選ばないことを意味する 0-枝を持つ．終端接点は 0 か 1 の値を持ち 0 はそのアイテム集合が存在しないことを示し，1 は存在することを示す．

集合 $S = \{\{a, b\}, \{a, c\}, \{c\}\}$ を ZDD で表したものを図 1 に示す．例として，a を選び (1-枝)，b を選ばず (0-枝)，c を選ぶ (1-枝) と終端接点 1 に辿り着きアイテム集合 $\{a, c\}$ が存在するということが分かる．一方 a を選び (1-枝)，b を選ばず (0-枝)，c も選ばない (0-枝) と終端接点 0 に辿り着きアイテム集合 $\{a\}$ が存在しないということが分かる．

4. 既存手法

DPLL 手続きに基づく SAT ソルバーに ZDD を組み合わせて性能向上を図った既存研究がある [6]．多くの SAT ソルバーでは節をリスト形式で保持するが，この手法ではすべての節を ZDD により圧縮して保持する．そして ZDD 上での単位伝播を実現するため，ZDD を有向非巡回グラフ (DAG : Directed Acyclic Graph) に変換し，DAG 上で監視リテラルを実現している．

この手法では，CNF 式の 1 つの節を 1 つのアイテム集合とみなし，ZDD を構築する．例えば CNF 式 $F = (a \vee b \vee c \vee f) \wedge (a \vee b \vee \neg c \vee f) \wedge (a \vee b \vee \neg d \vee e) \wedge (a \vee \neg c \vee \neg d \vee \neg f) \wedge (\neg a \vee c \vee \neg d \vee \neg f) \wedge (b \vee \neg c \vee d \vee f)$ があったとき，これを ZDD に構築すると図 2 のようになる．ここで変数順序は $a < \neg a < b < \ldots$ であるとし，0-枝のないノードは 0-枝が終端接点 0 を指しているものとする．

次にこの ZDD を元に DAG を構築する．DAG を構築する

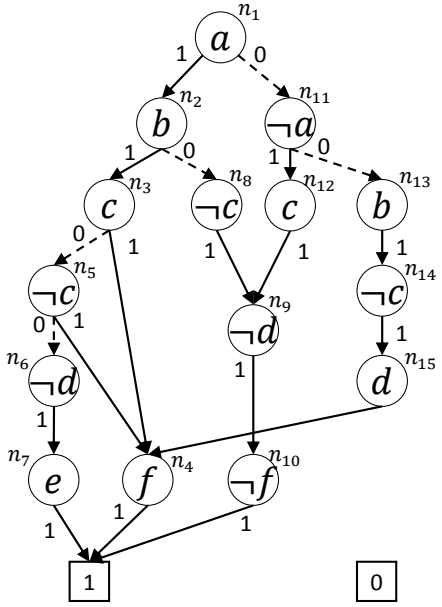

図 2: CNF 式 F から ZDD を構築

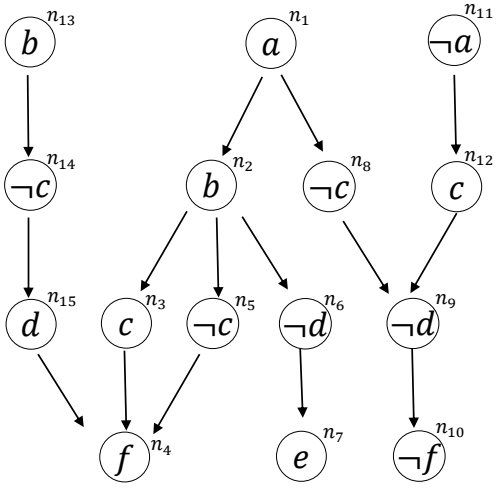

図 3: ZDD から DAG を構築

理由は探索の利便性のためである．DAG の各ノードは次のアイテムを指すノードのリスト *next_nodes* と，前のアイテムを指すノードのリスト *prev_nodes* を持つ．例えば図 2 のノード n_2 の *next_nodes* と *prev_nodes* はそれぞれ，$next\_nodes(n_2) = \{n_3, n_5, n_6\}$,$prev\_nodes(n_2) = \{n_1\}$ となる．図 2 の ZDD を元に DAG を構築すると図 3 のようになる．この時矢の根本から矢の先に向かう辺が *next_nodes* の要素で，矢の先から矢の根本への逆向きの辺が *prev_nodes* の要素を表す．

DAG を構築したら，この DAG 上のノードを監視することで単位伝播処理を実現する．監視中のノードを**監視ノード**と呼ぶ．まず DAG の各始点と終点を監視ノードとする．ここで始点とは *prev_nodes* の要素数が 0 のノードのことを指し，終点とは *next_nodes* の要素数が 0 のノードのことを指す．図 3 では始点が $\{n_1, n_{11}, n_{13}\}$ で，終点が $\{n_4, n_7, n_{10}\}$ となる．このように監視ノードを割り当てることで通常のリストによる監視リテラルと比較して，DAG 上の監視ノードではリテラルが共有されているため，監視ノードの数が監視リテラルに比べて

少なくなり，効率が良いことが分かる．最悪のケースでもリストによる監視リテラルと同等の監視ノードの数で済む．

そして探索中，監視ノードに偽が割り当てられたとき，監視リテラルの手法と同様に未割り当てノードが無いかを探す．偽が割り当てられた監視ノードを v としたとき，v の監視を未割り当てノードに移動させる必要がある．リストの場合はその節内のみを調べればよいが，DAG の場合は節のリテラルが共有されているので，$next\_nodes(v)$ を全て探索して未割り当てノードを探す必要がある．$next\_nodes(v)$ の各ノードを u としたとき，u が未監視の未割り当てノードであるならば監視を割り当て，u が偽ノードであるならば再帰的に未割り当てノードを探す．すべてに監視を割り当てることができたなら，v の監視を取り除く．

未割り当てのノードを探索する途中で，すべてのノードが偽であるパスや，1 つだけ未割り当てノードが有り他すべて偽ノードであるパスが見つかるときがある．前者は矛盾か単位伝播を引き起こす可能性があり，後者は単位伝播を引き起こす可能性がある．そのため $prev\_nodes(v)$ も探索を行う必要がある．こちらも同じように $prev\_nodes(v)$ を全て探索して未割り当てノードを探す．探索の途中で，すべてのノードが偽であるパスや，1 つだけ未割り当てノードが有り他すべて偽ノードであるパスを見つけたら，$next\_nodes(v)$ の探索結果と合わせて矛盾を引き起こしたのか，単位伝播を引き起こしたのか判断し，それぞれ処理を行う．

5. 拡張にあたっての問題点

前述のアルゴリズムを基に CDCL 版に拡張するにあたって，問題となることが 4 つ存在する．

1 つ目はこのアルゴリズムが提案された時と比べて，現在の SAT 問題の大きさは飛躍的に増大しているということである．既存手法が考案された 2002 年頃の SAT 問題の大きさは，SAT Competition 2002 を例に挙げると最もサイズが大きい問題でも変数数が約 30 万，節数も約 90 万程度であったが，SAT Competition 2014 では変数数が約 1400 万，節数が約 5000 万の問題がある．2002 年と 2014 年の SAT Competition の問題の大きさの比較を図 4 に示す．このように近年の SAT 問題の大きさは飛躍的に増大しているので，既存手法では ZDD が巨大になり構築に多大な時間とメモリを必要としてしまう．場合によっては ZDD を構築することすらできない場合もある．

2 つ目は CDCL には DPLL にはなかった学習節があるということである．学習節は矛盾が発生するたびに生成されるので非常に数が多い．学習節をどのように取り扱うのかを考える必要がある．

3 つ目は節学習のために単位伝播の理由をどのように保持するかということである．従来のリスト形式では節へのインデックスを保持しておけばよかったが，DAG ではノードへのインデックスを保持してもどの節を指しているのかが分からないためインデックスを保持するだけではうまくいかない．

4 つ目は DAG 上で監視リテラルを行なう際，監視ノードの数が増大してしまうということである．ある程度の増大は仕方がないが無駄な監視ノードが多ければ多いほどソルバーの性能が低下してしまうので，できるだけ無駄な監視ノードを割り当てなくても済むような手法が必要になる．

6. 提案手法

1 つ目の ZDD が巨大になるのを解決するために，CNF 式から 1 つの巨大な ZDD を構築するのではなく，CNF 式をある節数で分割し分割された CNF 式に対してそれぞれ小さな ZDD を構築し複数の ZDD を構築することで全体の CNF 式を表す．

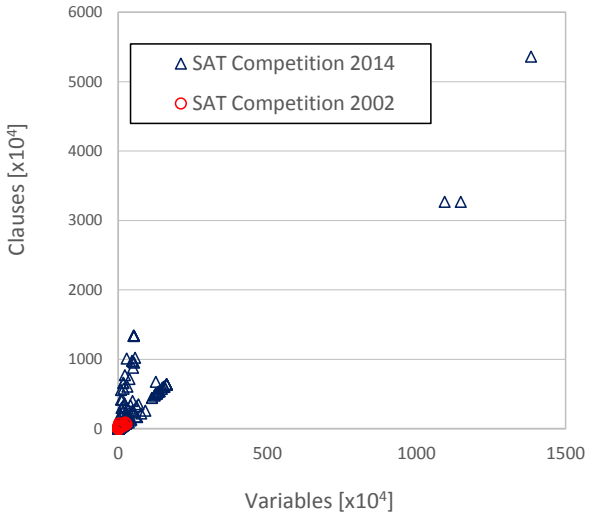

図 4: SAT competition の問題の大きさの比較

2 つ目の学習節の取り扱いに関して，学習節は探索中に追加と削除を繰り返し行っている．しかし学習節が追加されるたびに ZDD を構築していては時間が掛かってしまう．さらに追加された学習節のおよそ 9 割は探索中に破棄されてしまう．そのため ZDD に学習節を追加しても，学習節がすぐに破棄され ZDD からも破棄しなければならないということになる．そこで今回は学習節を従来通りリストで管理するものとする．これについては 8 章の今後の課題で改善案を述べる．

3 つ目の単位伝播の理由の保持に関しては，単位伝播処理を行なう際に単位節のパスを見つけたらそのパスをスタックに積み，節として参照できるようにする．この手法ではメモリを多く使用してしまうため今後の課題として改善策を検討していく．

4 つ目の監視ノードの数が増大してしまうことを抑えるために，監視ノードに色を付けて区別する手法を提案する．既存手法は監視ノードに色は無かったが，提案手法では監視ノードに赤と黒の色を付けて区別できるようにする．監視ノードの初期配置時，始点には赤色の監視，終点には黒色の監視を割り当てる．このように割り当てることで DAG の各パス上には赤と黒の監視がそれぞれ 1 つだけ存在していることになる．DAG の各パス上には監視ノードが 2 つ割り当てられていればよいので，パス上に赤と黒色の監視がそれぞれ少なくとも 1 つあればよい．探索中，偽が割り当てられた監視ノードを v としたとき，$next\_nodes(v)$ の各ノードを探索し監視ノードを割り当てるが，このとき割り当てる監視ノードの色は v の色と同じものとする．探索中，監視ノード u の色が v の色と同じノードを訪れる場合がある．このときパス上には赤と黒色の監視ノードがそれぞれ少なくとも 1 つあるので，u を含むパス上に新たな監視ノードを割り当てる必要はない．これによって監視ノードの数の増大を抑えられることが期待できる．

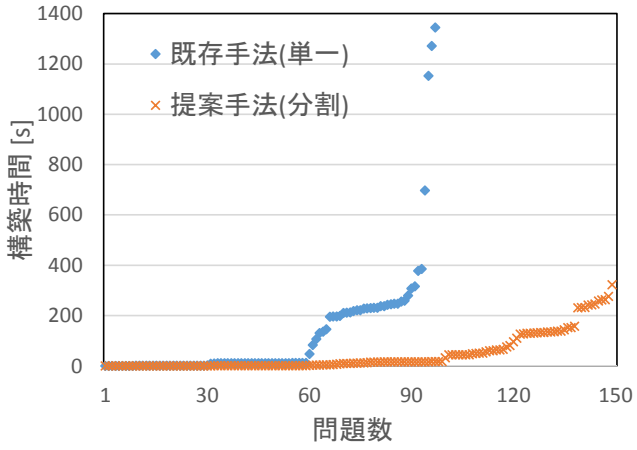

図 5: ZDD の構築時間

7. 性能評価と考察

代表的な CDCL ソルバーである MiniSat 2.2.0[7] に ZDD ライブラリの CUDD 3.0.0[8] を用いて，既存手法を実装を行なった．前述の提案手法の 4 つ目の監視ノードに色を付ける手法は未実装であり，その評価は今後の課題である．

性能評価として，CNF 式の分割による ZDD の構築時間の比較と，提案手法の求解数の比較を行う．CNF 式の分割による ZDD の構築時間の比較は，CNF 式を分割せず 1 つの ZDD を構築する従来の手法と，CNF 式を分割し複数の ZDD を構築する提案手法を構築時間で比較する．今回は分割数として先頭から 5000 節ごとに分割を行う．評価用問題として SAT Competition 2013 Application 部門の過去の問題を除いた 150 問を使用．実験環境は CPU Intel Xeon Processor E3-1230 V2 (3.30GHz)，メモリ 8GB を 6GB に制限，1 問あたりの制限時間は 3600 秒の環境で行なう．メモリをフルに使わず 6GB に制限したのは，スラッシングを防止するためである．

CNF 式の分割による ZDD の構築時間の比較の結果を図 5 に示す．縦軸は各問題の構築時間を示しており，横軸はある時間内に構築できた問題数を示している．この結果から既存の 1 つの ZDD だけで構築する手法では 90 問あたりで構築時間が飛躍的に増大してしまっているが，CNF を分割する提案手法ではすべての問題を構築することができ，構築時間が短縮できている．CNF 式を分割することで構築時間の向上は図れるが，分割することによって共有されるノードが減り，全体としてのノード数は増加することになる．そのため ZDD の構築時間と圧縮効果はトレードオフの関係にあり，分割数のより良い値を検討する必要があると考えられる．

次に提案手法を実装したソルバーと，基盤ソルバーの MiniSat との比較を行う．実験環境は前述の実験と同じである．提案手法の求解数の比較結果を表 1 に示す．結果から提案手法では性能が大きく低下していることが分かる．既存手法を最新のソルバーに実装するには，提案手法で述べた 3 つ目の単位伝播の理由の保持の効率化や，4 つ目の監視の数の増大を抑える手法の実装などしていく必要がある．

8. まとめと今後の課題

CDCL ソルバーでの ZDD 上での単位伝播処理を実現するために，既存手法の拡張を行った．既存手法をそのまま CDCL

表 1: 求解数

ソルバー	求解数 (SAT+UNSAT)
MiniSat 2.2.0	80 (36+44)
提案手法	57 (27+30)

ソルバーに導入しただけでは，大規模化した問題や学習節の取り扱いなどの問題があり，これらを解消する必要があることが分かった．大規模化した問題には CNF 式を分割し ZDD を構築することで，ZDD 構築の時間の短縮を図ることができることを示した．

今後の課題として，現在学習節はリスト管理されており，既存の CDCL ソルバーと学習節の処理が変わっていない．そこで学習節の中でも削除されない節というのが存在する [9] ので，そのような学習節を ZDD に取り込めるようにし差別化を図りたい．また ZDD 構築時，入力変数の順序付けや ZDD に追加する節の順序を変えることで，構築時間やメモリ使用量が大きく変わる．今現在は元の CNF 式の順序そのままであり工夫がないので，この手法に適した変数順序や追加する節の順序を検討する必要がある．また提案手法で述べた監視ノードに色を付ける手法が実装できていないので，これを実装し評価する．そして単位節，矛盾検出時の実装上の無駄な処理を効率化し，性能向上を図っていきたい．

参考文献

[1] Minato, S.: Zero-suppressed BDDs for set manipulation in combinatorial problems, In Proc. 30th ACM/IEEE Design Automation Conference, pp. 272-277 (1993).

[2] 湊真一: 超高速グラフ列挙アルゴリズム, p.177, 森北出版, (2015).

[3] 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).

[4] 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).

[5] 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).

[6] Aloul, F. A., Mneimneh, M. N. and Sakallah. K. A.: ZBDD-Based Backtrack Search SAT Solver, In International Workshop on Logic and Synthesis (IWLS) , pp. 131-136 (2002).

[7] Eén, N. and Sörensson, N.: An Extensible SAT-solver, in Proceedings of SAT-2003, pp. 502518 (2003).

[8] Somenzi, F.: CUDD: CU Decision Diagram Package (Online), <http://vlsi.colorado.edu/~fabio/>, (2016-3-14).

[9] Audemard, G. and Simon, L.: Predicting Learnt Clauses Quality in Modern SAT Solvers, in Proceedings of IJCAI-09, pp. 399404 (2009)