1D4-OS-11a-2in

乗法標準形で与えられた論理関数に対する二分決定グラフ構築の効率化

Efficient BDD Construction for Boolean Functions Given in CNF

岩下 洋哲 *1*2 Hiroaki Iwashita　戸田 貴久 *3 Takahisa Toda　津田 宏治 *4 Koji Tsuda　湊 真一 *2*1 Shin-ichi Minato

*1JST ERATO　*2北海道大学 Hokkaido Univ.　*3電気通信大学 UEC　*4産業技術総合研究所 AIST

We propose an algorithm to construct a binary decision diagram (BDD) for a Boolean function given as a CNF formula. With the maturing of SAT technology, techniques for encoding various constraint satisfaction problems into CNF formulas have been widely put into practical use, which are also useful as intermediate formats to obtain BDDs for real-life problems. In this paper, we describe a new technique to increase efficiency of a breadth-first search algorithm for BDD construction. Unlike modern SAT solvers, we pre-compute the rich information that minimizes backtracks using symbolic reachability analysis on transitions among sets of unsatisfied clauses. Experimental results show that our method is effective as a complement to conventional methods.

1. はじめに

SAT 技術の発展にともない、様々な制約問題を乗法標準形（CNF: Conjunctive Normal Form）の論理式に符号化する技術が広く実用化されている。CNF 式を充足する値割当て（充足解）の例は SAT ソルバーによって高速に求めることができるが、本研究では CNF 式の全ての充足解を圧縮表現した二分決定グラフ（BDD: Binary Decision Diagram）[Bryant 86] を構築する方法について議論する。BDD は実用上の多くの論理関数を効率良く取り扱うことができる。論理関数が BDD で表現されていると、充足可能性判定や等価性判定を含む様々なクエリに定数時間あるいは多項式時間で答えることができる。また、論理演算などの基本演算も効率的に実行することができる。全ての充足解を BDD で表現しておくことは、与えられた制約条件の下での最適化問題や数え上げ問題を繰り返し解きたいような場合に特に有用である。

論理関数と変数順序が同じであれば、最終的な BDD はその構築手順によらず同じになる。ところが計算時間や計算途中の BDD サイズは、その構築手順によって大きく異なることがある。最も素朴であり広く用いられている方法は、CNF 式の構造に従って各節に対応する BDD を構築し、それら全ての論理積を求める構築手順であろう。悪いことに、その計算途中では最終的な BDD よりもずっと大きな中間 BDD が現れることがある。そして、どのような中間 BDD が現れるかは論理積演算の適用順序にも依存している。これに対して Huang らは、充足可能性判定で有用な DPLL 手続きを応用した BDD 構築法を提案している [Huang 04]。素朴な構築法の方が速い場合もあるが、DPLL に基づく方法で著しい高速化と省メモリ化が達成される場合も少なくないことが報告されている。一方、戸田は CNF 式の構造を圧縮表現したゼロサプレス型三分決定グラフ（ZTDD）を構築した後にそれを BDD に変換する二段階の構築法を提案した [戸田 13]。この方法は、ほぼ全ての例において素朴な構築法よりも高速な BDD 構築を実現している。

もし仮に、入力となる CNF 式が大規模であるにも関わらずその充足解がほんの少数（あるいは充足不可能）である問題を対象とするのであれば、SAT ソルバー技術に基づくアルゴリズムが適していることは容易に想像できる。一方、素朴な構築法や ZTDD を用いた構築法では、そのような問題に対しても計算途中の BDD が大きくなってしまう可能性がある。しかしながら構築後の BDD を上記のように応用することを想定すると、我々が最も興味を持つ対象は明示的に列挙することが困難なほど多くの充足解を持つ CNF 式である。

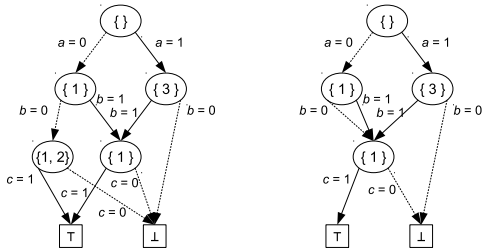
現在の多くの SAT ソルバーでは、変数への値割り当てに関する二分探索の中で単位伝播や節学習の処理を動的に行うことで探索空間を削減している。また、使用可能性の低い学習結果を削除して高速化とメモリ使用量削減をはかる処理なども含まれている。これに対して我々は、完全な探索結果のグラフ構造を BDD としてメモリ上に構築することを目的としているため、一般的な SAT ソルバーとはバランスの違った最適化を試みることにした。本文では、充足解へのパスを持つ状態空間を示した強力な条件を前処理の段階で計算しておくことで探索のバックトラックを回避するアルゴリズムを提案する。実験では素朴な構築法や ZTDD を用いた構築法との性能比較を行なう。

2. 節集合を状態とした探索アルゴリズム

例として CNF 式 $(a+c)(\bar{b}+c)(\bar{a}+b)$ を考え、a、b、c の順で変数の値を決定していくとする。単純な二分探索では $2^3=8$ 個の葉を持つ二分木を作ることになり、恒に変数の数に対して指数的な計算コストがかかる。これを改善するには、探索に適切な「状態」を導入して等価な状態を併合する必要がある。[Motter 02][Huang 04] には、まだ充足されていない節の集合を状態とすることで探索空間を定義した手法が示されている。$(a+c)$、$(\bar{b}+c)$、$(\bar{a}+b)$ をそれぞれ節 1、2、3 と呼ぶことにすると、最初のステップで変数 a の値を 0 に決めた後の状態は未充足節の集合を用いて {1,2}、1 に決めた後の状態は {2,3} と表現することができる。変数 a を含まない節 2 はこの段階では無関係であるため、省略して {1}、{3} と表現しても良い。図 1a に幅優先探索による BDD 構築の様子を示す。

我々は、この例で変数 a、b が決まった後の節 1 と節 2 は共にリテラル c だけを含むという意味で等価であることに着目し、さらなる状態空間の削減をはかった（図 1b）。一般に、複数の節に共通するサフィックスの部分で状態を融合できる。

連絡先: 岩下洋哲, JST ERATO 湊離散構造処理系プロジェクト, iwashita@erato.ist.hokudai.ac.jp



(a) 従来法　(b) 等価な節を考慮

図 1: 幅優先探索による BDD 構築

3. 論理関数処理による有効な探索領域の計算

前節の方法でトップダウンに BDD を構築していくことは可能であるが、最終的に充足解を示す集合 {} すなわち BDD の ⊤ 終端節点に到達できないパスを延々と計算し続けることは無駄である。そこで我々は、最終結果の BDD とは別に、節番号を論理変数とした BDD（あるいは ZDD[Minato 93]）によって {} に到達可能な状態（以後、「有効な状態」と呼ぶ）の集合を計算しておくことで、その正確な判定を可能にした。

有効な状態の集合は、変数値の決定順とは逆方向の探索によって求めることができる。図 1 の例において最終状態の一つ上のレベルで有効な状態は、c の値を適切に決めると最終状態 {} に遷移できる状態である。c に 0 を割り当てるとするなら {} でなければならず、1 を割り当てるとするなら {}、{1}、{2}、{1,2} のいずれかでなければならない。したがって有効な状態の集合は {{},{1},{2},{1,2}} である。

BDD を用いると、この計算を効率的に行うことができる。C_k を節 k が状態に含まれるか否かを示す論理変数とすると、最終状態の集合 {{}} に対応する論理関数は $\overline{C}_1\overline{C}_2\overline{C}_3$ であり、一つ上のレベルで有効な状態の集合を表現する論理関数は $\overline{C}_3$ である。一般に、あるレベルで有効な状態の集合を表現する論理関数 S からその一つ上の変数 v に対応するレベルの論理関数を求めるには、S においてリテラル $\overline{v}$ を持つ節の存在をドントケアとした論理関数とリテラル v を持つ節の存在をドントケアとした論理関数の論理和を計算すれば良い。同様に順方向に求めておいた到達可能状態集合で制約しながら計算しても良い。

4. 実験結果

BDD ライブラリとして CUDD を使用し、素朴な構築法、[戸田 13]、および提案手法を実装した。実験の結果を表 1 に示す。例題には ISCAS ベンチマーク回路の入出力関係を表現した CNF（c∗ と s∗）、SATLIB[*1] の問題の一部、およびドミノ敷き詰め問題 [Knuth 11] を Sugar[*2] で作成したもの（domino-∗）を用いた。変数の静的順序付けには MINCE[Aloul 01][*3] を用いている。CPU 時間の上限は 10,000 秒、メモリ使用量の上限は 64GB に設定した。

5. おわりに

CNF 式から BDD を構築する問題において、充足解へのパスを持つ状態空間を示した強力な条件を前処理の段階で計算しておくことで探索のバックトラックを回避するアルゴリズムを提案した。実験では素朴な構築法や ZTDD を用いた構築法との性能比較を行なった。本手法は既存手法よりも効率が良い場合も悪い場合もあるが、どちらの場合も大きな性能差が現れることがある。したがって、それらは互いに補完し合う手法として有効であるといえよう。

∗1 http://www.satlib.org/
∗2 http://bach.istc.kobe-u.ac.jp/sugar/
∗3 http://www.aloul.net/Tools/mince/

表 1: 実験結果

	ナイーブ法		ZTDD 法		提案手法	
	Time (sec)	Mem (MB)	Time (sec)	Mem (MB)	Time (sec)	Mem (MB)
c432	10.8	1750	3.3	1700	3.1	594
c499	20.6	1762	3.1	1732	4.3	678
c880	*timeout*		1965.5	42162	791.5	11017
c1355	105.9	1995	9.6	2081	8.7	693
s510	0.9	1594	0.3	1595	5.9	703
s526	0.8	1588	0.1	1584	0.4	557
s526n	0.4	1583	0.1	1581	0.5	563
s641	5.8	1639	0.8	1637	1.6	576
s713	2.9	1620	0.7	1618	1.4	574
s820	*timeout*		*memout*		42.0	1985
s832	*timeout*		*memout*		42.0	1884
s953	32.5	1910	12.2	2007	97.6	2797
s1196	343.2	5764	106.0	5945	41.5	1274
s1238	76.5	2421	18.3	2244	67.7	2183
s1423	871.9	6470	139.2	5904	108.7	1612
s1488	*timeout*		*timeout*		4993.7	59237
s1494	*timeout*		*memout*		714.0	14250
aim-100-1_6-no-1	3644.6	59119	1466.1	48960	31.5	2182
aim-100-1_6-no-2	734.0	19915	224.0	11932	0.1	526
aim-100-1_6-yes1-1	20.1	2020	8.7	2013	0.1	525
aim-100-1_6-yes1-2	13.4	2074	5.7	1905	0.1	528
aim-100-2_0-no-1	0.0	1573	0.0	1574	*timeout*	
aim-100-2_0-no-2	*memout*		*memout*		258.0	10191
aim-100-2_0-yes1-1	0.1	1577	0.0	1578	232.4	9719
aim-100-2_0-yes1-2	3.8	1783	1.8	1712	124.5	5341
ais8	2.5	1616	0.5	1598	3794.2	49913
ais10	*timeout*		462.6	11854	*memout*	
bw_large.a	80.9	1830	16.8	1726	47.1	2299
huge	119.0	1787	16.6	1721	65.2	3213
par16-1	2346.7	3687	169.1	3376	539.2	14398
par16-1-c	849.8	4481	178.3	4126	142.1	5983
par16-2	*timeout*		*memout*		354.9	13267
par16-2-c	3395.7	16946	591.2	11483	136.1	5682
par16-3	243.7	1802	11.9	1768	3269.3	56443
par16-3-c	24.9	1639	3.2	1619	595.2	16369
domino-15	52.2	1747	3.1	1706	0.2	555
domino-16	149.7	2010	7.2	1856	5.3	645
domino-17	489.0	2701	27.4	2358	0.9	597
domino-18	866.0	3657	49.8	2863	25.3	963
domino-19	1860.9	6310	81.2	4151	0.8	611
domino-20	6057.9	11034	322.2	7647	136.3	2424

参考文献

[Aloul 01] Aloul, F. A., Markov, I. L., and Sakallah, K. A.: Faster SAT and Smaller BDDs via Common Function Structure, in *Proceedings of the 2001 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '01, pp. 443–448 (2001)

[Bryant 86] Bryant, R. E.: Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Transactions on Computers*, Vol. 100, No. 8, pp. 677–691 (1986)

[Huang 04] Huang, J. and Darwiche, A.: Using DPLL for efficient OBDD construction, in *In Seventh International Conference on Theory and Applications of Satisfiability Testing, SAT 2004, Revised Selected Papers*, pp. 157–172 (2004)

[Knuth 11] Knuth, D. E.: *The Art of Computer Programming, Volume 4A, Combinatorial Algorithms, Part 1*, Addison-Wesley Professional, 1st edition (2011)

[Minato 93] Minato, S.: Zero-suppressed BDDs for Set Manipulation in Combinatorial Problems, in *Proceedings of the 30th Design Automation Conference*, DAC '93, pp. 272–277 (1993)

[Motter 02] Motter, D. and Markov, I.: A Compressed Breadth-First Search for Satisfiability, in *Algorithm Engineering and Experiments*, Vol. 2409 of *Lecture Notes in Computer Science*, pp. 29–42, Springer Berlin Heidelberg (2002)

[戸田 13] 戸田貴久 : 論理関数の CNF から BDD の効率的な構築法, 情報処理学会研究報告. AL, アルゴリズム研究会報告, Vol. 2013, No. 3, pp. 1–8 (2013)