知能と複雑系　125－1
（2001. 7. 23）

二分決定グラフによるモデル生成木の刈込み

岡 雄一郎† 越村 三幸‡ 長谷川 隆三‡

† 九州大学大学院システム情報科学府
‡ 九州大学大学院システム情報科学研究院

〒816-8580 福岡県春日市春日公園 6-1
{oka,koshi,hasegawa}@ar.is.kyushu-u.ac.jp

あらまし　論理関数を計算機上で表現する方法として二分決定グラフ (BDD) があり、論理演算や恒偽、恒真性の判定を効率よく計算できるとされている。これをモデル生成法に基づく一階述語定理証明システム MGTP に組み込むことにより MGTP が生成するモデル生成木を刈り込むことができる。本研究では、実際に BDD を組み込んだ MGTP を実装し、従来の MGTP と性能を比較した。

キーワード　モデル生成法，二分決定グラフ

Pruning Model Generation Proof Tree with Binary Decision Diagrams

Yuuichirou Oka　Miyuki Koshimura　Ryuzo Hasegawa

Graduate School of Information Science and Electrical Engineering
Kyushu University

6-1 Kasuga-Kouen, Kasuga, Fukuoka 816-8580, Japan
{oka,koshi,hasegawa}@ar.is.kyushu-u.ac.jp

Abstract　Binary Decision Diagram(BDD) is a data structure that expresses Boolean expressions on computers. We can effectively manipulate Boolean expressions and determine their satisfiability with BDDs. We can enhance proving power of MGTP (Model Generation Theorem Prover) by pruning proof tree of MGTP using BDDs. We implement MGTP with BDD, and compare it with standard MGTP.

key words　Model Generation, Binary Decision Diagram

1 はじめに

MGTP(Model Generation Theorem Prover)[2]は、モデル生成法に基づく一階述語論理の定理証明システムである。モデル生成法は、節集合のエルブランモデルの構築を試みることで、節集合の充足可能性の判定を行う。構築途中のモデルをモデル候補と呼ぶ。モデル生成法は、モデル候補で充足されない節の基礎例を用いてモデル候補を拡張していく。この拡張は、全ての節が充足されるか、矛盾が導かれるまで続けられる。一つのモデル候補が複数のモデル候補に拡張されることがあり、これは証明の分岐に相当している。

拡張は、盲目的に行われるので、証明には貢献しない拡張や分岐後の重複証明といった無駄な推論が行われる危険がある。本稿では、これらの無駄な推論を二分決定グラフ (BDD:Binary Decision Diagram) を利用して回避する手法を提案し、その評価を行う。BDD は命題論理式の表現法であり、BDD に対する論理演算は計算機上で効率的に実現できる。

無駄な推論の回避は次の原理に基づいて行われる：

> 現在着目しているモデル候補が、それ以前までに用いられた基礎例の集合に矛盾していれば、そのモデル候補に対する拡張は不要である。

なぜなら、このモデル候補を含むようなモデルは存在し得ないので、拡張を続けていっても、いずれ矛盾が導かれるからである。ここで、基礎例の集合もモデル候補も命題論理式として表現できるので、この矛盾検査は、(原理的には) 命題論理式の充足不能性の検査となる。本稿では、この充足不能性検査を BDD を用いて行う。

以下では、まず BDD と MGTP の概説を行う。次いで、MGTP 証明への BDD 演算の組み込み法について述べる。最後に、本手法の実験結果を述べ、それに対して考察する。

2 二分決定グラフ BDD

2.1 BDD のデータ構造

BDD とは、論理関数をコンパクトに表現できる非巡回有向グラフである。BDD の各節点は一つの論理関数を表し、

$$f_v = \bar{v} \bullet f_{v0} + v \bullet f_{v1}$$

と書くこともできる。ここで f_v は変数 v でラベル付けされた論理関数 f で節点に相当する。f_{v0}, f_{v1} は、0 枝、1 枝の指す子節点である。すなわち、BDD は論理関数の Shannon 展開をグラフで表現したものとも言える。

ここでは BDD を 図 1 のように表す。これは $a \vee \bar{b}$ を表す BDD である。BDD 中の円は節点 (変数節点,node) と呼ばれ、変数でラベル付けされている。正方形で囲まれた 0,1 は特に定数節点と呼ばれる。それぞれ *false*,*true* に対応するものである。節点から左下、右下に出た線をそれぞれ “0 枝”,“1 枝” といい、変数が 0,1 の値を持つ場合に対応する。今回は変数がモデルに含まれる場合に、正リテラルならば 1 枝を、負リテラルならば 0 枝を対応させることにする。

また枝が指す先の節点を子節点、その逆を親節点と呼ぶことがある。図 1 の場合、節点 n0 は節点 n1 の親節点であり、n1 は n0 の子節点である。

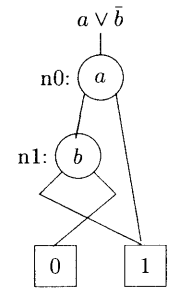

図 1: BDD of $a \vee \bar{b}$

また、先頭にある節点 (図 1 では n0) を 根 (root) と呼ぶ。根から任意の値をとって終端の “0” または “1” へたどる筋をここでは 路 (path) と呼ぶ。図 1 の場合、3 本の路が存在する。すなわち、

- $a \rightarrow 1$
- $\bar{a} \rightarrow \bar{b} \rightarrow 1$
- $\bar{a} \rightarrow b \rightarrow 0$

である。

一般に BDD と言えば、既約で順序つきのものを指す。

「既約」というのは、冗長であったり、等価であったりする、不必要な節点を取り除いたという意味である。

冗長な節点というのは、図 2 の左のような場合である。ここで節点 n0,n1 は節点 n2 の親節点であり、節点 n3 は n2 の子節点である。n2 について見てみると、0 枝、1 枝のどちらも同じ節点 n3 を指しているので、n2 にラベル付けされた変数の値によらず、n3 へたどりつくことになる。従って、n2 は冗長な節点であり、

n2 を取り除き、n2 を指していた枝を n2 が指していた n3 を指すようにしても等価である。

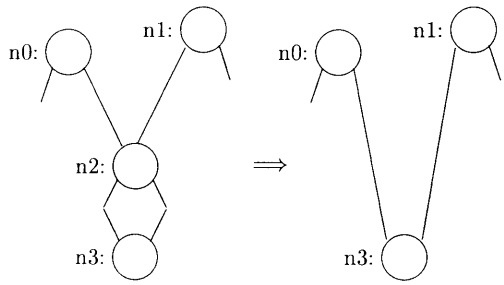

図 2: 冗長な節点の除去

等価な節点というのは、図 3 のような場合である。節点 n5 と節点 n6 は同じ変数でラベル付けされ、0 枝が節点 n7、1 枝が節点 n8 とそれぞれ同じ子節点を指している。従って、節点 n4 の n6 を指す枝を n5 に付け替え、n6 を取り除いても、BDD は等価である。

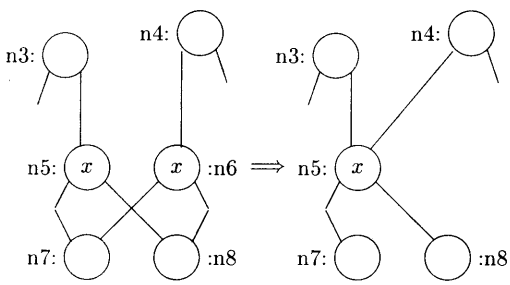

図 3: 等価な節点の除去

「順序つき」というのは BDD 中に出てくる変数それぞれに異なる優先順位がついていて、その優先順位に従って BDD が構成されているという意味であり、任意の path をたどった場合、その path に現れる変数の順序はこの優先順位に等しくなっている。ただし、path 上に入力されたすべての変数が存在するとは限らない。一部の path には現れない場合や、BDD そのものから除去されてしまう場合もあり得る。

二つの論理式の既約で順序つきの BDD が等しいならば、二つの論理式は等価である。

2.2 BDD の構成法

論理式 A を表す BDD はその部分論理式の BDD から A の主論理記号に従って構築される。

図 1 を例にとると、図 4 のような手順となる。左上から順に b, $\bar{b}$, a, $a \vee \bar{b}$ を表す BDD である。はじめ

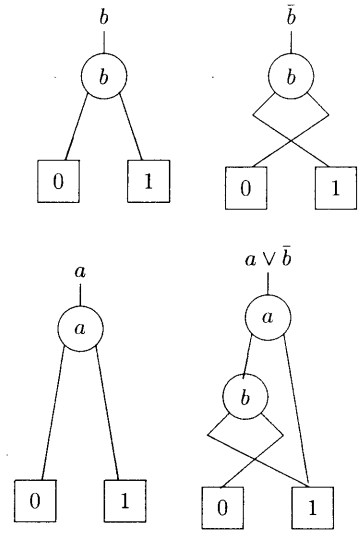

図 4: apply 演算 $(a \vee \bar{b})$

に b を表す BDD に否定演算を行って、$\bar{b}$ の BDD を作る。次に a と $\bar{b}$ との論理和演算を行うことによって、$a \vee \bar{b}$ を表す BDD を求めることができる。

これは、否定や論理和演算に限らず、論理積演算の場合でも適用できる。このように、BDD に対して論理演算を行って新たな BDD を得る演算は apply 演算と呼ばれる。

3 モデル生成法

モデル生成法において、節は次のように含意式の形で表現される。

$$A_1 \wedge A_2 \wedge \ldots \wedge A_n \rightarrow C_1 \vee C_2 \vee \ldots \vee C_m$$

ここで、$A_i (1 \leq i \leq n)$ および $C_j (1 \leq j \leq m)$ は原子論理式 (atom) である。$\rightarrow$ の左側を前件部、右側を後件部という。$n = 0$ のとき、前件部を特に $true$ と書き、正節と呼ぶ。一方 $m = 0$ のとき、後件部を特に $false$ と書き、負節と呼ぶ。それ以外の節は混合節と呼ぶ。また、節 $A_1 \wedge A_2 \wedge \ldots \wedge A_n \rightarrow C_1 \vee C_2 \vee \ldots \vee C_m$ が基礎アトムの集合 M に置換 σ のもとで violated であるとは、$\forall i (1 \leq i \leq n) A_i\sigma \in M \wedge \forall j (1 \leq j \leq m) C_j\sigma \notin M$ であることをいう。

モデル生成法は、与えられた節集合に対するエルブランモデルを、空集合から始めて構成的に求める証明手法である。

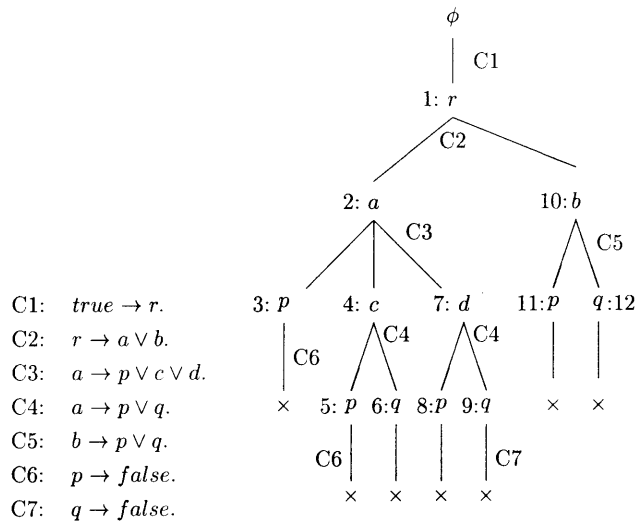

図 5: 例題 S1 とそのモデル生成木

モデル生成法には次の二つの規則がある。規則中 $\mathcal{M}$ は構成途中のモデルの集合を表し、各要素をモデル候補と呼ぶことにする。$\mathcal{M}$ の初期値は $\{\phi\}$ である。

- モデル拡張規則: 混合節もしくは正節 $A_1 \wedge A_2 \wedge \ldots \wedge A_n \rightarrow C_1 \vee C_2 \vee \ldots \vee C_m$ が、あるモデル候補 $M \in \mathcal{M}$ に置換 σ のもとで violated である時、M の代わりに各 $C_j\sigma$ を M に加えてモデル候補を拡張する ($\mathcal{M} := \mathcal{M} \cup \{M \bigcup_{j=1}^{m} \{C_j\sigma\}\} \backslash \{M\}$)。
- モデル棄却規則: 負節 $A_1 \wedge A_2 \wedge \ldots \wedge A_n \rightarrow false$ が、あるモデル候補 $M \in \mathcal{M}$ に置換 σ のもとで violated である時、M を棄却する ($\mathcal{M} := \mathcal{M} \setminus \{M\}$)。

二つの規則のいずれも適用できなくなった時点で、節集合の全モデルが $\mathcal{M}$ の要素として得られる。モデル生成法は健全で完全であることから、$\mathcal{M} \neq \phi$ ならば節集合は充足可能であり、$\mathcal{M} = \phi$ ならばその節集合が充足不可能であることがわかる。

なお MGTP では、入力節に値域限定という制約を設けている。値域限定というのは、後件部に現れる変数は、全て前件部に現れるという制約である。これによりモデル候補の各要素は、基底であることが保証される。

図 5 は C1 から C7 の 7 つの節で構成される問題 S1 と、その問題に対して MGTP が生成するモデル生成木である。各アトムの左 (12 のみ右) にある数字は MGTP が生成したモデル候補につける番号で、本文中の M_n の n と対応している。例えば 7 番であれば、$M_7 = \{r, a, d\}$ のモデル候補を表す。

ここで、根からリテラルをたどったものを枝と呼ぶ。枝が閉じるとは、枝上のリテラルの集合において矛盾が発生するか、モデル棄却規則が適用される場合を言う。完成した枝とはこれ以上モデル拡張規則が適用できない枝か、閉じた枝のことである。一般に枝といった場合には完成した枝のことを指す。

まず、空集合 $\mathcal{M} = \{\phi\}$ から始める。ϕ に $C1$ を用いてモデル拡張規則を適用することにより、$M_1 = \{r\}$ が得られる。次に $C2$ により M_1 から $M_2 = \{r.a\}$ と $M_{10} = \{r, b\}$ が得られる。M_2 は $C3$ によって $M_3 = \{r, a, p\}$,$M_4 = \{r, a, c\}$,$M_7 = \{r, a, d\}$ に拡張される。ここで、M_3 は $C6$ によって棄却される。M_4 は $C4$ によって $M_5 = \{r, a, c, p\}$ と $M_6 = \{r, a, c, q\}$ に拡張されるが、それぞれ $C6$,$C7$ によって棄却される。M_7 も同様である (M_8,M_9)。一方、M_{10} は $C5$ によって $M_{11} = \{r, b, p\}$ と $M_{12} = \{r, b, q\}$ に拡張されるが、それぞれ $C6$,$C7$ によって棄却される。今や、$\mathcal{M} = \phi$ となりどの規則も適用できないので S1 は充足不能であると結論できる。この場合の枝数は 7 でいずれも閉じている。また MGTP が生成したモデル候補数は 12 である。MGTP は、深さ優先、左優先でモデル候補を拡張していく。

4 BDDを利用したモデル生成木の刈り込み

4.1 MGTPへのBDDの組み込み

今回試作したMGTP(以下従来のMGTPと区別する必要がある場合はBDD-MGTPと言う)では、従来のMGTPの動作に加えて、新たに以下のことを行う。

4.1.1 BDDの拡張

$$A_1\sigma \wedge A_2\sigma \wedge \ldots \wedge A_n\sigma \rightarrow C_1\sigma \vee C_2\sigma \vee \ldots \vee C_m\sigma$$

は、前件リテラルを後件に移動させ、

$$true \rightarrow C_1\sigma \vee C_2\sigma \vee \ldots \vee C_m\sigma \vee \bar{A}_1\sigma \vee \bar{A}_2\sigma \vee \ldots \vee \bar{A}_n\sigma$$

としても等価である。このことから、節から論理関数

$$C_1\sigma \vee C_2\sigma \vee \ldots \vee C_m\sigma \vee \bar{A}_1\sigma \vee \bar{A}_2\sigma \vee \ldots \vee \bar{A}_n\sigma$$

を得ることができる。

このことから、節を適用してモデル候補を拡張する際、その節から上記のようにして得られた論理関数をBDDで表し、MGTPが保持するBDDとAnd演算を行うことによってBDDを拡張する。

4.1.2 BDDによる恒偽判定

モデル候補を拡張した際、BDDを調べることによって生成されたモデル候補が恒偽であるかどうかを判定する。恒偽となることが判定できれば、そのモデル候補を棄却することができる。具体的には、節点にラベル付けされたアトムについて、着目しているモデル候補の中の対応するリテラルが正ならば1枝を、負ならば0枝をたどる。対応するリテラルがない場合は両方の枝をたどる。この規則に従う全ての路が定数節点"0"にたどり着くならば、そのモデル候補は恒偽であると判定される。1つでも定数節点"1"にたどり着いたならば現時点では恒偽ではないと判定する。

また1つのモデル候補が複数のモデル候補に拡張された場合、MGTPではそのうち1つを着目するモデル候補として保持し、残りをスタックに保存する。一つの枝が完成した場合にこのスタックからモデル候補を取り出すが、その際にもこのモデル候補を生成したモデル候補について同様の恒偽判定を行う。

いずれの場合にも、この判定作業はBDDが負節によって拡張されるまでは行われない。

4.2 BDD-MGTPの動作例

先程の例題S1で説明する。なお、以下の説明中に単にBDDという場合には、MGTPが保持するBDDを指し、BDD拡張時に一時的に作られる、適用する節を表すBDDとは区別する。BDDの図は表現を簡易にするため、等価な定数節点を複数描いている。変数の優先順位については、今回試作したBDD-MGTPでは問題に出てきた順に優先するようにしているので、それに合わせている。例題で言えば、最初に出てくるrは最も優先順位が高く、次いでa、b、c、最後に出てくるdは最も優先順位が低くなる。

以下で使うモデル候補番号は図5と共通にしているが、BDD-MGTPでは生成に至らないモデル候補があるため、実際にBDD-MGTPがつけるモデル候補番号とは異なる。

1. BDDの初期値は1である。
2. まず、空集合ϕに$C1$を用いてモデル拡張規則を適用しようとする。ここでは、$C1$は初めて適用されるので、$C1$を表す仮のBDDを作成し、BDDとのAnd演算が行こなわれる。現時点でBDD $=true$なので、仮のBDDがそのままBDDとなる。$M_1 = \{r\}$が得られる。
3. 次に$C2$を適用しようとする。$C2$より論理関数$\bar{r} \vee a \vee b$が得られる。これを表す仮のBDDを作り、BDDとAnd演算を行い、BDDを拡張する。M_1と$C2$から$M_2 = \{r, a\}$と$M_{10} = \{r.b\}$が得られる。
4. $C3$によってBDDは拡張され、M_2を$M_3 = \{r, a, p\}$.$M_4 = \{r, a, c\}$, $M_7 = \{r, a, d\}$に拡張する。始めにM_3に注目し、M_4,M_7はスタックへ積む。
5. $C6$によってBDDは図6のように拡張され、ここでM_3はMGTPによって棄却される。BDDが負節によって拡張されたため、これ以後のモデル候補拡張時には常にBDDによる恒偽判定が行われる。
6. M_2について恒偽判定を行うが現時点では恒偽と判定できないので、M_4をスタックから取り出し拡張する。始めに$C4$を利用してBDDを図7に拡張する。
7. $C4$とM_4より$M_5 = \{r, a, c, p\}$と$M_6 = \{r, a, c, q\}$に拡張する。M_5についてBDDを調べるとr –

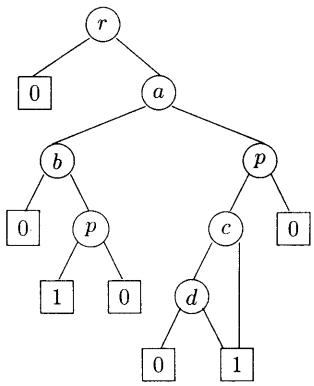

図 6: 手順 5 終了時点

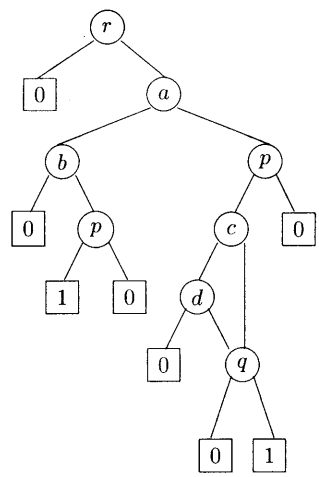

図 7: 手順 6 終了時点

$a-p-0$ より恒偽と判定され、棄却される。M_4 は現時点では恒偽とは判定されないので、スタックから M_6 を取り出す。$C7$ によって BDD は図 8 に拡張され、M_6 は MGTP によって棄却される。

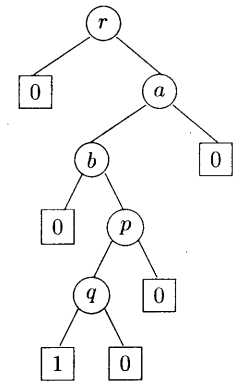

図 8: 手順 7 終了時点

8. M_2 について BDD を調べると $r-a-0$ より恒偽であると判定される。よって、スタック中の M_7 はモデル候補となることなく棄却され、M_8,M_9 へは拡張されない。

9. M_1 の恒偽判定を行うが現時点では恒偽とは判定できないので、スタックから M_{10} を取り出し、BDD を $C5$ により拡張すると BDD=0 となる。M_{10} と $C5$ より $M_{11} = \{r, b, p\}$ に拡張し、$M_{12} = \{r, b, q\}$ はスタックに積む。ここで BDD を調べると恒偽なので、M_{11} は棄却される。スタック中の M_{12} についても同様にモデル候補となることなく棄却される。

10. 今や、$\mathcal{M} = \phi$ となりどの規則も適用できないので例題は充足不能であると結論できる。

図 9 は、BDD-MGTP におけるモデル生成木で、生成されたモデル候補数は 8、棄却された枝の数は 4 である。このように MGTP がモデル候補を拡張する前に、モデル候補や枝を棄却することにより、モデル生成木を刈り込み、証明時間を短縮できる可能性があると考えられる。

5 実験

5.1 方法

Java により実装した BDD-MGTP と従来の MGTP に、それぞれ問題を解かせ、その性能を比較した。問

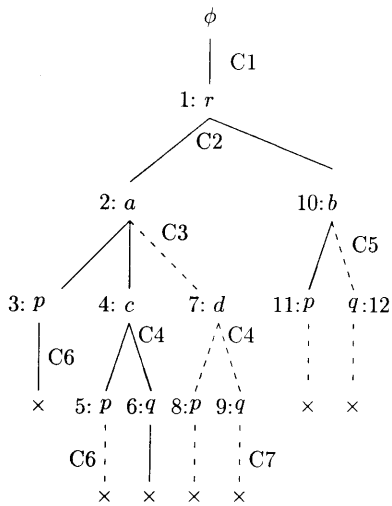

図 9: BDD-MGTP による S1 に対するモデル生成木

題は Thousands of Problems for Theorem Provers[3] から、143 問について計測した。

計測に用いたマシンのスペックは，以下の通りである．

CPU：Intel PentiumII 400MHz
メモリ容量：256MB
OS：Microsoft Windows 2000

従来の MGTP として JavaCMGTP Version 32g を、BDD-MGTP は BD_CMGTP Version 32g V7 を、Java は JDK1.3.0 を使用した。

なお、今回は SAT/UNSAT の判別のみを行う方式で計測した。従って、モデルが見つかる問題の場合はそのモデルが見つかった時点で証明は終了する。また、実験を自動で行うため、Perl の子プロセスとして JavaVM を起動するという方法を採用した。Perl は ActiveState Tool Corp. の ActivePerl v5.6.0 for MSWin32-x86-multi-thread Binary build 623[1] を利用した。

5.2 考察

まず始めに、両方式で解けた問題について、2 つの観点から考察を行う。ここでは、各データを CMGTP の場合を 1 としたときの BDD-MGTP の割合の平均を示す。表 1 の下段は、BDD の効果が特に発揮される SYN009 を除いた場合である。

[1] http://www.activestate.com/Products/ActivePerl/

表 1: 両方式で解けた場合の比較
(下段は SYN009 を除く)

	Least	Average	Max
Failed branches	0.000152	0.730	1
Total atoms	0.000474	0.648	1
Proving time	0.130	206.430	4995
Failed branches	0.00574	0.740	1
Total atoms	0.00644	0.657	1
Proving time	1	209.256	4995

5.2.1 探索空間

棄却された枝数 (Failed branches) と生成されたモデル候補数 (Total atoms) を比較することで考察する。

BDD の効果が顕著に表れる SYN009 では、CMGTP では 19683 本の枝を棄却し、29526 個のモデルを生成したのに対して、BDD-MGTP では 3 本、14 個にまで激減した。それ以外の問題は、枝数、モデル数ともに 1/200 程度に減少した問題からまったく変わらない問題が存在した。枝数に関しては平均すると 73%に、UNSAT の問題では 64%に、またモデル数に関しても平均 65%に、UNSAT の問題では 57%に、減少したことから、BDD は探索空間を小さくすることに関しては有効に働くと言える。

5.2.2 証明速度

証明時間 (Proving time) を比較することで考察する。

前の項目と同様に、SYN009 では、CMGTP よりも早く判別することができた。しかしながら、SYN009 を除くと最速の場合でも CMGTP と同じ時間であり、5000 倍の時間がかかるものも存在した。平均でも 200 倍強となっている。

この理由として、BDD は命題論理しか扱うことができず、述語論理などで atom の数が増大すると BDD が大きくなりやすく、その演算と標準化に時間がかかることが考えられる。

この他に、CMGTP に比べて、メモリ不足による終了が多く (表 5.2.2)、特に、CMGTP が判別できて、BDD-MGTP が判別できなかった (異常終了した) 場合の原因は、JavaVM が原因を報告しなかった場合を除くと全てメモリ不足によるものであった。これは BDD を拡張する際に、最悪の場合、節点数が指数関数的に増加し、かつその節点数に比例してメモリの使用量が増えるためと考えられる。

表 2: 両方式で解けた問題の判定結果による比較
(上段:UNSAT(52 問)/下段:SAT(23 問))

	Least	Average	Max
Failed branches	0.000152	0.635	1
Total atoms	0.000474	0.569	1
Proving time	0.13	166.657	2747
Failed branches	0.745	0.955	1
Total atoms	0.522	0.826	1
Proving time	2.5	294.620	4995

表 3: 結果別の問題数

	BDD	MGTP
successful	75	109
OutOfMemoryError	46	17
Timeout (600sec.)	10	15
StackOverflowError	6	1
Exception	6	1

6 おわりに

本研究では、Java による BDD-MGTP の試作を行い、MGTP との比較実験によりその性能を確かめた。

比較実験では，BDD を利用することにより、モデル生成木を刈込む点については効果があったが、BDD を作成するために非常に時間を費やすため、証明時間が著しく増加することも判明した。また、節点数に比例してメモリの使用量が増加し、メモリ不足による異常終了も比較的多かった。

ただし、以下の点を考慮することにより，さらなる速度向上、メモリ使用量の節約を図ることも可能であると考えられる。

- 節点数の抑制

  BDD の演算は、BDD の節点数を m,n とすると、$\mathcal{O}(mn)$ の時間を必要とする。従って、BDD の節点数を減らすことによって、演算に要する時間を減らすことができる。また、節点数に比例してメモリを使用するため、メモリ使用量の節約を図ることもできる。具体的には、以下の 2 点を挙げておく。

  - 述語論理に関しての BDD の拡張

    今回試作した BDD は命題論理に限定されており、atom の数が増えるにつれて節点が増大し、その演算に非常に時間を費やしたが、BDD が述語論理を扱えるようになれば、atom が増加しても、節点数を抑えられ、演算に費やす時間とメモリ使用量を減らすことができると考えられる。

  - 変数の優先順序の最適化

    BDD は変数の優先順位を変更することによって、節点数も変化することが知られている。従って、最小の節点数で BDD を構成することのできる最適な変数の優先順位を見つけることによって、節点数を減らすことができる。しかし、そのアルゴリズムとしては、$\mathcal{O}(n^2 3^n)$ (n:入力変数の数) のものしか発見されておらず、多くの変数を扱うことができない。今回、変数の優先順位については問題節に登場した順としたが、よりよいアルゴリズムを採用することによって変数の優先順序を最適化することにより、節点数を減らすことも可能であると考えられる。

- 適当な標準化を施した BDD の採用

  今回試作した BDD は「既約な順序つき」の BDD をつくるような仕様となっている。しかしながら、MGTP にとっては BDD は目的ではなく、手段の一つであるため、完全に「既約な順序つき」にする必要はなく、同じ程度の精度が得られるならば、むしろ適切な程度の「既約な順序つき」にすることによって、演算に費やす時間を減らすことができると考えられる。

今回開発した BDD は、あくまで試験的なものであり、上に挙げた改良点を含め、改良の余地は多々存在する。これらの改良による BDD の拡張や演算の高速化とメモリ使用量の節約、及びその評価を今後の課題とする。

参考文献

[1] 石浦菜岐佐. BDD とは. 情報処理 Vol.34 No.5 pp.585-592, May 1993.

[2] 長谷川隆三, 藤田博. Java によるモデル生成型定理証明系 MGTP の開発. 情報処理学会論文誌 Vol.41 No.6 pp.1791-1798, June 2000.

[3] Geoff Sutcliffe and Christian Suttner. Thousands of problems for theorem provers, v2.3.0, November 1999. http://www.cs.jcu.edu.au/~tptp/.