

## 二分決定グラフによるモデル生成木の刈込み

岡, 雄一郎

九州大学大学院システム情報科学府知能システム学専攻 : 修士課程

長谷川, 隆三

九州大学大学院システム情報科学研究所知能システム学部門

越村, 三幸

九州大学大学院システム情報科学研究所知能システム学部門

<https://doi.org/10.15017/1515817>

---

出版情報 : 九州大学大学院システム情報科学紀要. 8 (1), pp.49-54, 2003-03-26. 九州大学大学院システム情報科学研究所

バージョン :

権利関係 :



## 二分決定グラフによるモデル生成木の刈込み

岡 雄 一 郎\* ・ 長谷川隆三\*\* ・ 越 村 三 幸\*\*

### Pruning Model Generation Proof Tree with Binary Decision Diagrams

Yuuichirou OKA, Ryuzo HASEGAWA and Miyuki KOSHIMURA

(Received December 13, 2002)

**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 the proving power of MGTP (Model Generation Theorem Prover) by pruning proof tree of MGTP using BDDs. In this paper, we propose a method which postpones updating BDD in order to suppress memory consumption and compare its performance with standard MGTP and a previous version of MGTP with BDDs.

**Keywords:** Model generation, Binary decision diagram(BDD), Automated theorem proving, Proof tree

#### 1. は じ め に

MGTP(Model Generation Theorem Prover)<sup>1)</sup>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。我々は以前、二分決定グラフ(Binary Decision Diagram, BDD)を利用して回避する手法を提案した<sup>4)</sup>。BDDは命題論理関数の表現法であり、BDDに対する論理演算は計算機上で効率的に実現できる。ここでは生成木の刈込みには効果があるものの、メモリの使用量や証明時間の点で悪化することが分かった。そこで本稿では、BDDの構築手順を見直し、メモリの使用量や証明時間を改善する手法を提案し、その評価を行う。

以下では、まずBDDとMGTPの概要を説明した後、MGTP証明へのBDD演算の組み込み法と今回提案する手法を示す。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<sup>2)</sup>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、

$$f_v = \bar{v} \cdot f_{v=0} + v \cdot f_{v=1}$$

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。  $f_{v=0}, f_{v=1}$  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。

本稿ではBDDを **Fig. 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。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。

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。**Fig. 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。

また、先頭にある節点(**Fig. 1** ではn0)を根(root)と呼ぶ。根から任意の値をとって定数節点へたどる経路をここでは路(path)と呼ぶ。**Fig. 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。

**Fig. 2** の左図において、節点n0,n1は節点n2の親節点

平成14年12月13日受付

\* 知能システム学専攻修士課程

\*\* 知能システム学部

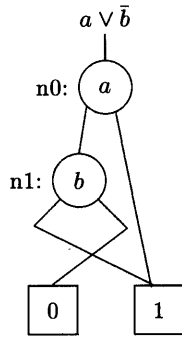


Fig.1 BDD for  $a \vee \bar{b}$ .

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、n2にラベル付けされた変数の値によらず、n3へたどりつるので、n2は値の決定に関与していないことになる。このような節点n2を冗長な節点と呼ぶ。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なBDDとなる。

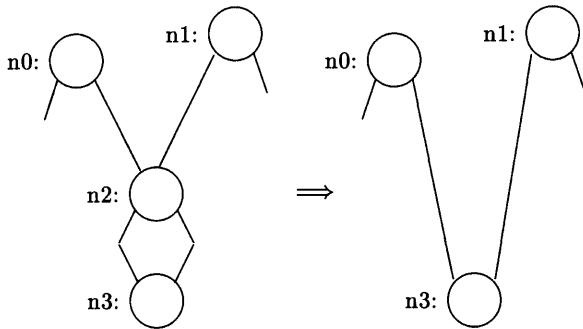


Fig.2 Eliminating redundant nodes.

Fig. 3の左図の場合、節点n6と節点n7は同じ変数でラベル付けされ、0枝が節点n8、1枝が節点n9とそれぞれ同じ子節点を指している。このような節点n6、n7を等価な節点という。この場合、節点n5のn7を指す枝をn6に付け替え、n7を取り除いても、BDDは等価である。

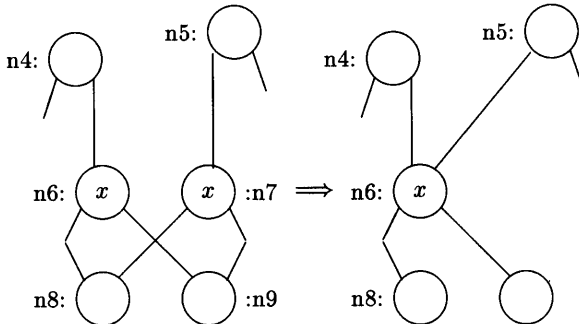


Fig.3 Eliminating equivalent nodes.

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BDDを既約なBDDと呼ぶ。

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が構成されている場合、順序付きのBDDと呼ぶ。順序付きのBDDにおいて、根から任意の路をたどった場合、その路に現れる変数の順序関係はこの優先順位の関係と同じになっている。ただし、任意の路をたどった時に入力されたすべての変数が出現するとは限らない。一部の路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。

一般にBDDと言えば、既約で順序付きのものを指す。変数の優先順位が等しいならば、入力の順序をどのように変えても、既約なBDDは同じものとなる。

### 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。

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

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。 $m = 1$ のとき、Horn節と呼び、Horn節でない節はNon-Horn節と呼ぶ。また、節 $A_1 \wedge A_2 \wedge \dots \wedge A_n \rightarrow C_1 \vee C_2 \vee \dots \vee C_m$ が基礎アトム集合Mに置換 $\sigma$ 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。

モデル生成法には次の二つの規則がある。規則中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。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  $A_1 \wedge A_2 \wedge \dots \wedge A_n \rightarrow C_1 \vee C_2 \vee \dots \vee C_m$  が、あるモデル候補  $M \in \mathcal{M}$  に置換  $\sigma$  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 ( $\mathcal{M} := \mathcal{M} \cup \{M \cup \bigcup_{j=1}^m \{C_j \sigma\}\} \setminus \{M\}$ ).
- モデル棄却規則: 負節  $A_1 \wedge A_2 \wedge \dots \wedge A_n \rightarrow false$  が、あるモデル候補  $M \in \mathcal{M}$  に置換  $\sigma$  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

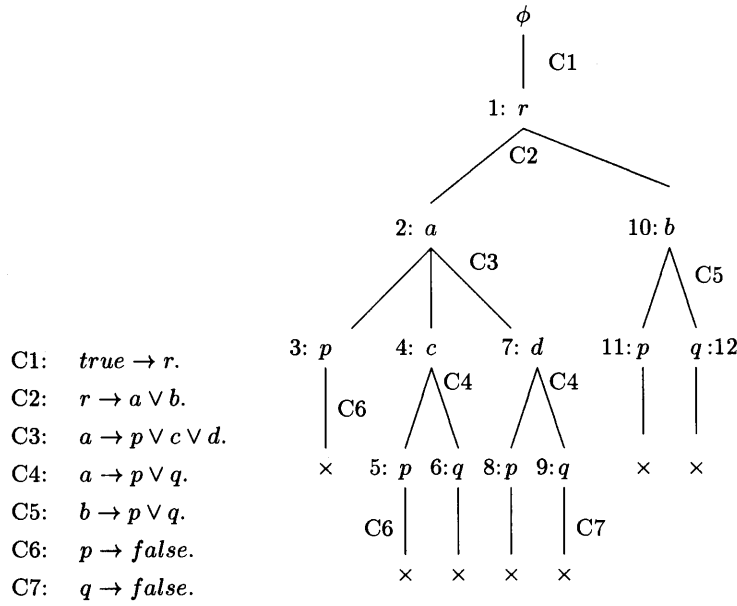


Fig.4 Example clause set S1 and its proof tree.

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。

Fig. 4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(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。 $\phi$ 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。 $\mathcal{M} = \phi$ 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。

## 4. BDDを利用したモデル生成木の刈込み

### 4.1 MGTPにおけるBDDの役割

モデル生成法は健全で完全な定理証明法である。しかしながら、盲目的に拡張を行うため、貢献に関与しない無駄な拡張や、同じ部分証明木を複数生成する重複証明といった無駄な推論を行ってしまう可能性がある。

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。

### 4.2 MGTPへのBDDの実装

BDDを組み込んだMGTP(以下BDD-MGTPと呼ぶ)では、従来のMGTPの動作に加えて、新たに以下のことを行う。

#### 4.2.1 BDDの拡張

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

は、前件リテラルを後件に移動させ、

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

としても等価である。このことから、節から論理関数

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

を得ることができる。

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、BDD-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。なお、BDD-MGTPが保持するBDDの初期値は1である。

#### 4.2.2 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、そのモデル候補をこれ以上拡張することなく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に正リテラルが含まれていれば1枝を、負リテラルが含まれていれば0枝をたどる。対応するリテラルが含まれてい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、ある問題節により1つのモデル候補 $M_{org}$ 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後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 $M_{org}$ について同様の恒偽判定を行う。この条件によるBDD検査で棄却された場合には、これ以上 $M_{org}$ を拡張してもモデルは存在しないことができる。これは問題節が証明に貢献していない場合に対応する。

いずれの場合にも、この判定作業はモデル候補が負節によって拡張されるまでは行われぬ。

### 5. btmup BDD-MGTP

以前の研究<sup>4)</sup>では、BDDの演算・構築に要する時間やメモリの要求量が非常に大きくなってしまいう問題が発生した。本稿ではこれらの点を改善するため、次のような改善案を提案し、BDD-MGTPに実装した。以下、この版のBDD-MGTPをbtmup BDD-MGTPと呼ぶ。

#### 5.1 BDDの構築手順の変更

BDD-MGTPでは、問題節を利用すると同時にBDDに組み込んでいたが、実際には証明に関与していない節があることが分かっており、様々な研究がなされている<sup>3)</sup>。

そこで、BDDの拡張を当該節の証明終了後に遅らせることによって、実際には証明に関与しなかった節によるBDDの拡張は行わないようにした。

具体的には、

- モデル候補の拡張に使われた問題節と拡張前のモデル候補を対にしてスタックに積んでおく。ただし、その問題節が正節ならば即座にBDDを拡張する。この理由については次節で述べる。
- 枝が閉じた場合、スタックから積んでおいた問題節と対応するモデル候補を取り出し、そのモデル候補とBDDを用いて恒偽判定を行う。ここで恒偽と判定されれば、取り出した問題節は証明に関与していないとわかるので、この問題節を利用したBDDの拡張は行わない。恒偽と判定されなければ、BDDを拡張する。この動作はまだ棄却されていないモデル候補が残っている問題節を取り出すまで繰り返される。

#### 5.2 正節によるBDD拡張

正節は定理証明において、前提条件を表すものであり、モデル生成法では証明はここから始まる。正節がHorn節ならばモデル候補には正節の後件のリテラルが必ず含まれる。これはBDDにおいてこのリテラルが対応するノードの0枝をたどることは無いということである。一方、5.1節の方式で正節による拡張を即座に行わない場合、その正節は証明の最後までBDDに取り込まれないため、該当するノードの0枝も演算の対象となり、BDDの拡張の際、ほとんどの場合、無駄な演算が行われる。また、その分BDDのノード数も増加する。正節がNon-Horn節の場合でもHorn節の場合より無駄な演算が少なくなるとはいえ、同じことが言える。

BDDの構築に関しては、2つのBDDの2項演算はBDDのそれぞれのノード数を $m, n$ とすると、 $O(mn)$ の手間がかかる。また、メモリの使用量はBDDのノード数にほぼ比例する。従って、メモリの使用量の点からも、BDDの構築に要する計算量の点からも、BDDのノード数を削減することは重要である。

また、MGTPにおいてBDDを拡張する際にはand演算のみを利用するので、定数節点0に到達する路を増やすことで、BDDの拡張に関する演算を短縮することができる。ほか、その後の拡張においてもノード数を抑制することができる。

以上の理由から、5.1節の通り、正節に関しては即座にBDDの拡張を行うことにした。

**Table 1** Statistics : Averages on MGTP=1.

	BDD	btmup
# of Failed Branch	0.730	0.771
# of Total Atoms	0.648	0.677
Proving Time	207.8	153.3

**Table 2** Performance Comparison of BDD-MGTP and btmup BDD-MGTP.

	Ave.	Min.	Max.
# of Failed Branch	1.037	0.982	2.000
# of Total Atoms	1.017	0.933	1.329
Proving Time	0.967	0.181	6.488
# of Created Nodes	0.863	0.0288	2.940
# of Retracted Nodes	0.773	0.0973	3.012
# of Maximum Nodes	0.928	0.0288	2.840
# of BDD Updates	0.862	0.0545	1.233

### 5.3 BDDにおける変数の優先順序

BDDのノード数は変数の優先順序に非常に敏感に変化する。以前の研究<sup>4)</sup>では証明に登場した順に優先順位を決定し、最初に出てきた変数が優先度が最も高くなるようにした。Fig. 4の例題S1であれば、 $r$ が最も高く、 $a, b, p, c, d, q$ の順序である。その後、後件については後ろから優先順位を決定する( $r, b, a, d, c, p, q$ )方式で実験を行ったが、最初の方式よりノード数が増大した。そこで今回は最初の方式を基本に、(1)証明に登場した節から優先順位を決定する、(2)BDDの拡張に利用する節から優先順位を決定する、2つの方式を事前に実験し、全般的に結果が良好であった(1)の方式を採用した。

## 6. 性能評価, 実験

### 6.1 概要

Javaにより実装したBDD-MGTP, btmup 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。問題はThousands of Problems for Theorem Provers<sup>6)</sup>から、143問について計測した。

計測では、従来のMGTPとしてJavaCMGTP Version 32gを、BDD-MGTPはBD\_CMGTP Version 32g V7を、btmup BDD-MGTPはBDD btmup v0.06 rev02を、JavaはJava2 SDK1.4.1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。

Table 1は、各データをMGTPの値を基準とした相対値の平均を取ったものである。表中のBDDはBDD-MGTPを、btmupはbtmup BDD-MGTPを示す。

**Table 3** Result.

	MGTP	BDD	btmup
Successful	109	75	74
Time Out(600sec.)	15	10	31
Stack Overflow	1	6	1
Out of Memory	17	46	35
Exception	1	6	2

Table 2は、各データをBDD-MGTPの値を基準とした相対値の平均と、その最小値、最大値に関する表である。

Table 3は、証明の終了原因ごとの問題数である。MGTPのみで解ける問題が34問、btmup BDD-MGTPのみで解けるのが2問、MGTPとBDD-MGTPで解けるのが3問、すべての方式で解けるのが72問である。

なお、MGTPよりも速く判定できる問題数は、BDD-MGTPでは1問のみであったのに対し、btmup BDD-MGTPでは3問である。

### 6.2 BDDの拡張回数

Table 2の# of BDD Updatesを比較すると、平均で86%に減少しており、今回採用した方式は有効に働いていると言える。

### 6.3 証明速度

証明時間(Proving time)を比較することで考察する。Table 2によると、BDD-MGTPに対して3%程度の改善が見られる。また問題数で比較すると、72問中44問で早くなっており、今回採用した方式は証明速度の向上にも寄与している。しかしながら、Table 1のとおり、MGTPとは平均153倍となり、BDD-MGTPよりも改善しているとはいえ、まだ大きな隔たりが見られる。

### 6.4 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。Table 2によると、数%程度数値が増えており、BDDによる棄却精度が落ちていることを示している。これはBDDに組み込む必要が無いと判定した問題節がその後で有効に働く場合に、BDD-MGTPではすでに組み込まれているので棄却できるものの、btmup BDD-MGTPでは棄却できない場合が考えられる。Table 1でも若干の悪化が見られるが、いずれの比較結果も証明速度の向上からすると納得できる範囲だと考える。

### 6.5 BDDの作成効率

Table 2の作成ノード数(# of Created Nodes)、破棄ノード数(# of Retracted Nodes)、最大ノード数(# of Maximum Nodes)で比較する。平均すると77%から

93%に減少しているが、問題数で見ると最大ノード数は減少38問,増加31問で、今回の方式は改善に若干寄与している、という程度である。ただ、Table 3 から分かるように、メモリ不足による終了が減少しており、メモリ使用量についてはある程度減少しているのではないかと推測される。

## 7. ま と め

本稿では、メモリ使用量の削減や、BDDの構築時間の削減を目的として、MGTPにおけるBDDの構築手順を変更したbtmup BDD-MGTPを提案し、その実証を行った。

実験の結果、各項目で若干の改善は見られたものの、従来のMGTPには及ばなかった。

BDDの最大規模は、変数順位や節の投入順序に非常に敏感に変化する。これに関連してさまざまな研究<sup>5)</sup>がなされているものの、MGTPにおけるBDDの構築は、MGTPの動作と密接に関連しており、最適な設定を見つけることは今後の課題である。

5.2節で述べた全体に関与しやすい問題節を優先するという方式は、今回の改良とあいまって、かなりの改善効果をもっており、これを負節にも適用することにより、更なる改善をすることが可能であると考えられる。

## 参 考 文 献

- 1) 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.
- 2) 石浦菜岐佐. BDDとは. 情報処理学会論文誌 Vol.34 No.5 pp.585-592, May 1993.
- 3) 越村三幸, 長谷川隆三. 証明の依存性解析による定理証明の冗長探索の削除. 人工知能学会誌 Vol.15, No.6, 2000.
- 4) 岡雄一郎. 二分決定グラフを利用したJava-MGTPの試作. 九州大学工学部電気情報工学科卒業論文, February 2001.
- 5) 奥乃博, 湊真一. 二分決定グラフによる制約充足問題の解法. 情報処理 Vol.36 No.8 pp.1789-1798, August 1995.
- 6) Geoff Sutcliffe and Christian Suttner. Thousands of Problems for Theorem Provers, v2.5.0, July 2002. <http://www.tptp.org/>.

