二分決定グラフを用いた数独パズルの解探索と列挙
Enumerating Solutions of Sudoku Puzzle Using BDDs

立石 匡 † 湊 真一 †
Kyou Tateishi Shin-ichi Minato

1. はじめに

制約充足問題とは，与えられた制約条件を満たす入力パタンの組合せを探索する問題であり，社会の様々な局面で現れる基本的な問題である．制約充足問題を論理式で記述し，論理関数のコンパクトな表現法である二分決定グラフ (BDD:Binary Decision Diagram)[1] を用いて，解を求める方法 [2] が提案されている．

本稿では，制約充足問題の一例として数独パズルをとりあげ，BDD を用いて解探索と列挙を行う方法について述べる．数独（ナンバープレイス）はペンシルパズルの一種 [3] であるが，近年，世界的なブームとなっている．数独の制約条件を論理式として記述し，それを満たす論理関数を表現する BDD を構築することができれば，条件を満たす解の個数をチェックしたり，ヒントの有無が問題の難易度に与える影響を解析することが容易になる．奥乃らは，我々と連携し，数独問題の難易度定義と問題自動生成法を開発するプロジェクトを進めている [4]．また，渡辺らのグループは，対称解を除く全ての解に一意なインデックスを付与し，これを高速に求める方法を提案している [5]．

数独の解集合を表す BDD の大きさは，ヒントの数や配置にも依存するが，一般には非常に大規模なサイズのグラフとなる．本論文では，BDD を構築する際に，記憶量や計算時間を節約するための論理式の記述方法や，その効果について報告する．

2. 数独パズルの制約条件

数独とは，図 1 のように 9 × 9 の行列の各マスに，1〜9 の数字を埋めるパズルであり，
・各行で数字が重複しない．
・各列で数字が重複しない．
・3 × 3 の各ブロック内で，数字が重複しない．
という 3 つのルールを満たさなければならない．通常，ヒントとしていくつかのマスの数字が与えられる．ヒントの個数は 24〜32 程度が一般的であり，ヒント数が少ないと問題が難しくなる．ヒント数が少な過ぎると解が一意に定まらなくなる．一意に解が定まる最小のヒント数として，現在のところヒント数が 17 個のパタンが知られている．

4		3		1				2
	7			2			3	
5				3		8		
	5			4	2		6	
6	9		1	5	7		4	3
	4		9	6			1	
		2		7				5
	6			8			9	
8				9		1		7

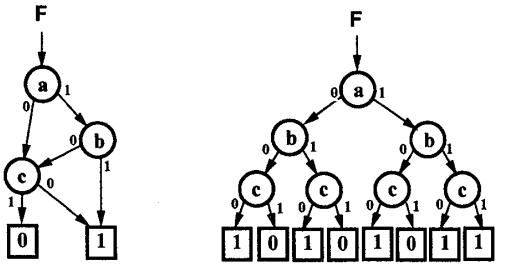
(a) 問題

4	8	3	7	1	6	9	5	2
1	7	9	5	2	8	4	3	6
5	2	6	4	3	9	8	7	1
3	5	1	8	4	2	7	6	9
6	9	8	1	5	7	2	4	3
2	4	7	9	6	3	5	1	8
9	1	2	3	7	4	6	8	5
7	6	5	2	8	1	3	9	4
8	3	4	6	9	5	1	2	7

(b) 解答

図 1: 数独パズルの例

†北海道大学 大学院 情報科学研究科 アルゴリズム研究室



(a) BDD. (b) 二分決定木.

図 2: BDD と二分決定木: $F = (a \wedge b) \vee \bar{c}$.

3. 二分決定グラフ (BDD)

BDD は，図 2(a) に示すような論理関数のグラフによる表現である．これは，論理関数の値を全ての変数について場合分けした結果を図 2(b) のように二分木グラフで表し，これを縮約することにより得られる．このとき，場合分けする変数の順序を固定し，(1) 冗長な節点を削除する，(2) 等価な節点を共有する，という処理を可能な限り行うことにより「既約」な形が得られ，論理関数をコンパクトかつ一意に表せることが知られている．

BDD は，多くの実用的な論理関数を比較的少ない記憶量で一意に表現することができる．また，2 つの BDD を入力とし，それらの二項論理演算 (AND,OR 等) の結果を表す BDD を生成するアルゴリズム [1] が知られており，データが計算機の主記憶に収まる限りは，その記憶量にほぼ比例する時間内で論理演算を効率よく実行できるという特長を持つ．

我々は BDD 処理系として，算術論理式計算プログラム BEM-II[2] を用いた．BEM-II は論理式を簡単に扱う電卓のようなシステムである．入力スクリプトとして単純な論理式だけでなく，算術演算を含む論理式が許される．算術演算を使用すると，例えば変数 a, b, c のうちただ 1 つだけが真であるという択一型制約条件を

```
F = (a + b + c == 1)
```

の様に簡潔に記述できる．この式は BEM-II により

```
F = (a&!b&!c)|(!a&b&!c)|(!a&!b&c)
```

と等価な論理関数を表す BDD が生成される．ただし "!" は否定，"&" は論理積，"|" は論理和を表す．この様に，BEM-II では算術論理式を用いて，問題を簡単にコーディングし，BDD を生成することができる．

4. 論理式による制約条件の記述

各マスごとに，数字に対応する 9 個の論理変数を用意する．変数名は，1 文字目を行 (上から順に a〜i)，2 文字目を列 (左から順に a〜i)，3 文字目を数字 (1〜9) として，各マスに aa1〜aa9 の様に 9 個ずつ割当てる．例えば aa4 は，左上隅のマスの数字が 4 のときに 1(真)，それ以外は 0(偽) の値をとる．論理変数は最大で 729 個となるが，ヒントが与えられているマスには論理変数を割当てないことにする．これにより，ヒント 1 つにつき論理変数を 9 個ずつ減らすことができ，総計算量の削減に繋がる．

以下に数独の制約条件の論理式を列挙する．

(1) 各マスに数字は 1 つしか入らない

```
aa1 + aa2 + aa3 + ··· + aa8 + aa9 == 1
```

ヒントの無いマスごとに，この条件を設ける．

(2) 各行で数字が重複しない

```
aa1 + ab1 + ac1 + ··· + ah1 + ai1 == 1
```

の様に，各行毎に 9 個の条件式が作られる．ヒントが与えられている場合，例えば図 1(a) の 1 行目であれば

```
0 + ab1 + 0 + ad1 + 1 + af1 + ag1 + ah1 + 0 == 1
```

の様に，そのマスにその数字のヒントがあるなら 1 を，他の数字があるなら 1 を記述する．

(3) 各列で数字が重複しない

```
aa1 + ba1 + ca1 + ··· + ha1 + ia1 == 1
```

となる．ヒントがある場合，(2) と同じ様に記述する．

(4) 各ブロック内で数字が重複しない

```
aa1 + ab1 + ac1 + ba1 + bb1 + ···  + cc1 == 1
```

この様に，部分ブロック毎に照合する．ヒントがある場合は (2)，(3) と同じように様に記述する．

一般に，BDD では論理変数の順序を変えると，グラフの形が変わりノード数にも大きな変化が生じる場合がある．変数順序に関しては今後検討の余地はあるが，本稿では上位から順に aa1〜aa9，ab1〜ab9，ac1〜ac9，···，ii1〜ii9 の順で固定した．この順序は関係が深い変数同士が近い位置にあるので，比較的良いと考えられる．

以上に示した各制約毎に BDD を作成し，全ての論理積を順に計算していく．全ての論理積をあらわす BDD が構築できたとき，それが数独の解集合となる．計算過程で BDD は増大していくが，終盤は解の候補が絞られて減るため，最終 BDD ノード数は途中よりも小さくなる．

4.1 制約条件の演算順序の工夫

論理変数の順序づけが同じであっても，制約条件の演算の順序を変えることで，計算途中の BDD の形が変わるので，計算の効率が変わる場合がある．そこで，マス→行→列→ブロックの順に計算する方法と，マス→行→ブロック→列の順に計算する方法の 2 通りを比較した．前者を手順 1，後者を手順 2 と呼ぶことにする．

また変数順序の下位の変数から順に論理式に登場するようにすると，計算途中の BDD の高さが比較的低い状態での計算が多くなるので，グラフをたどる時間が節約できると期待される．具体的には行・ブロックの条件を計算する過程で，ii9〜ii1，ih9〜ih1，···，aa9〜aa1 の順でそれらの変数を含む条件式を計算する．さらに各過程の計算を分割 (3 行→ 3 ブロック，計 3 セットに分ける) して，徐々に変数を登場させ，最後に列計算を aa〜ii の順で計算する．以下，この手法を手順 3 と呼ぶ．

5. 実験結果と考察

本実験で使用した PC は，AMD Opteron 885, 2.6GHz, Red hat 64 Linux (gcc 3.4.5) の 64 ビット機であり，128Gbyte の単一の実メモリ空間を 1 プロセスで占有使用できる．この PC で主記憶上に構築可能な BDD の大きさは約 30 億ノードで，これは汎用 PC による BDD 処理系としては，現時点で世界最大級のものである．

実験で用いた例題は，市販の問題集 [3] から，ヒント数 18〜35 の範囲で 7 題選択した．

5.1 各手順の計算時間の比較

手順 1〜3 のそれぞれについて BDD を作成する時間を比較した．なお，論理変数の順序は同じなので，最終的に得られる BDD は全て同一である．

まず，手順 1 と手順 2 との実行時間の差が顕著であることがわかる．計算途中の様子をみると，列条件の計算を始めた途端に BDD のノード数が爆発的に増大しており，早い段階で列計算を行う手順 1 が不利であると見られる．一方，手順 3 は手順 2 よりもさらに数倍程度，速度を改善できることがわかった．問題によって改善率にばらつきが見られるのは，ヒントの配置に因ると思われる．ノード数が急増する前に有効なヒントに出会えた場合に，計算時間は少なくて済むからである．

問題 NO.	ヒント数	最終 BDD ノード数	実行時間 (秒) 手順 1	手順 2	手順 3
6	35	414	16.47	0.52	0.30
7	33	432	26.04	0.49	0.24
15	30	459	214.70	0.77	0.28
28	28	477	745.32	1.20	0.54
59	24	513	42232.59	1.92	0.91
47	21	540	×	13.37	11.74
65	18	567	×	16.92	14.55

×:計算途中で主記憶あふれ

表 1: 各手順による計算時間

なお問題 No.47 と 65 においては，順序 1 では，途中でメモリあふれを起こし計算を完了できなかった．即ち，途中計算の BDD サイズが 30 億ノードを超えてしまったことを意味する．

5.2 ヒント数の削減と解集合の増加

一般に出題される数独は，解の盤面は唯一に定まるよう配慮されているが，さらにヒントを削ると複数個の解が得られるようになる．そこで問題 No.65 について，適宜ヒント数を削ることで，解がどの程度増えるかを調べたところ，下記のような結果となった．

17 ヒント:解　　9,434 個, ノード数　564,048
16 ヒント:解　188,105 個, ノード数　5,576,727
15 ヒント:解 3,379,627 個, ノード数 54,943,067

ヒント数を 14 まで削ると，現在のところ計算途中でメモリあふれを起こし，計算を完了できない状況である．

6. おわりに

本稿では，BDD を用いて数独の解集合を生成する際に，計算時間を削減するための工夫について述べた．実験の結果，論理変数の順序付けが同じであっても，制約条件の論理式の演算順序を工夫することで，ある程度の効果が得られることがわかった．

今後，さらなる BDD サイズの削減を目指して検討を行う．例えば，上下左右のヒントを事前に調べることにより，論理変数の数をさらに削減できる見込みがある．また，ヒントの位置に依存した変数順序づけ方法も検討課題である．今後も考察を進めていく予定である．

最後に，ご指導頂いた北大・ツォイクマン教授，ご助言頂いた京大・奥乃博教授に感謝致します．

参考文献

[1] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," Trans. Comput., C-35,8(1986),677-691.

[2] Shin-ichi Minato, "BEM-II: An Arithmetic Boolean Expression Manipulator Using BDDs," IEICE Trans. Fundamentals, Vol. E76-A, No. 10, pp. 1721-1729, Oct. 1993.

[3] ニコリ編著, "ニコリ「数独」名品 100 選," 文藝春秋社, 2006.

[4] 前田一貴, 奥乃博, "数独の問題作成支援システムの設計と開発," 情報処理学会全国大会, Mar. 2008.

[5] 戸神星也, 渡辺治, "数独インデックス公開プログラム," http://doorgod.org/sudoku/ .