2020 年度　卒　業　論　文

立体数独における
問題自動生成の高速化に関する研究

指導教員：渡辺 大地 教授

メディア学部 ゲームサイエンス
学籍番号　M0117151
鈴木 稔哉

2020 年 8 月

2020 年度　　卒　業　論　文　概　要

論文題目

立体数独における
問題自動生成の高速化に関する研究

メディア学部 **学籍番号：M0117151**	**氏名**	鈴木 稔哉	**指導教員**	渡辺 大地 教授

キーワード	立体パズル、立体数独、制約充足問題、バックトラック、自動生成

世界中で人気を誇るパズルゲームの 1 つに数独がある。一般的に、9 × 9 マスの盤面に、横一列、縦一列、太線で区切られたブロックのそれぞれで数字が被ることのないよう各マスに 1 から 9 の数字を当てはめるパズルである。数独に対しては様々な研究が行われている。初期配置に基づく問題の自動生成や、複数解の存在しない最小ヒント数を突き止める研究などがある。また、数独は上記で述べたもの以外に、サイズを変えたものや、ルールの条件を一部変えたものなど多くのバリエーションが存在する。このような数独のバリエーションの 1 つに、立体数独がある。これは、一般的な平面の数独に奥行の要素を追加し盤面を立方体にしたものである。ルールとして、盤面から得られる任意の平面の数独の盤面に対して数独のルールを満たさなければならない。立体数独は紙面上で遊ぶことが困難なため、アプリケーションが開発されたり、その UI を改良する研究が行われている。また、ルールも複雑化しマスの数も膨大になっていることから、問題を自動生成する研究も行われている。しかし、その研究では 4 × 4 × 4 のサイズの立体数独の問題自動生成しか確認されておらず、加えて提案手法では処理時間の観点から 9 × 9 × 9 のサイズの立体数独の問題自動生成は非現実的である。そこで本研究では、より高速な立体数独の問題自動生成を目的としてプログラムを実装、実験を行った。提案手法としては、深さ優先探索の一種であるバックトラックアルゴリズムを用いた。その結果、4 × 4 × 4 のサイズの立体数独の問題自動生成の高速化に成功した。また、9 × 9 × 9 の立体数独の問題自動生成も成功したが、膨大な時間を要するという課題も残った。

目 次

図 目 次

第 1 章

はじめに

1.1 研究背景と目的

数独とは、世界中で人気を誇るパズルゲームの 1 つである。この数独に関しては、問題を解く、問題を作成する、問題の特徴を探るといった研究が数多く行われている。例えば、解くことに関しては小河ら [1] による数独を解くソルバーの高速化を行った研究がある。問題作成に関しては前田ら [2] によって、問題作成時にヒントによって解を持つかどうかや解が一意に定まるかどうかを、UI で分かりやすく表示するツールの研究が行われた。また、小場ら [3] によって、数独の問題の難易度を判定するアプリケーションも研究されている。数独の特徴としては、McGuire ら [4] の研究によって、9 × 9 の盤面の数独において複数解を持たない最小ヒント数は 17 であることも判明している。さらに、数独はコンピュータ科学における研究分野としても盛んである。2016 年に座間ら [5] によって、初期配置を考慮した問題の自動生成を行う研究が行われた。数独は、ヒント数が少ない盤面を作成することが非常に困難であるが、この問題に対して那須ら [6] は、数独のヒント数のできる限り少ない問題をモンテカルロ木探索という探索アルゴリズムを用いて作成している。同様にヒント数の少ない問題の作成を、古川ら [7] がシミュレイテッドアニーリングという最適化問題を解く際の手法を用いて行っている。Timo ら [8] や、Amit[9] による、数独を多目的最

適化問題ととらえ、遺伝的アルゴリズムを用いて数独の問題を解く、作成する、評価するといった研究もある。松原 [10] は、人間が数独を解く際に用いる推論規則をコンピュータで実現することで、コンピュータによる問題の難易度判定や適切なヒントを出すことを目指した研究を行っている。大谷ら [11] は、当時パズルを解く手法として十分な成果がなかったポップフィールドネットワークを用いて数独の解を求めることに成功している。また、近年従来のパズルゲームを立体化したものの開発や研究が盛んである。例えば、$N \times N$ マスの盤面上に互いに攻撃しないように N 個の駒を配置する N クイーン問題や N ルーク問題の、盤面を立方体の表面上に拡張したもの [12] がある。ほかにも画面上にある数字の書かれたタイルをスライド操作によって足し合わせ、巨大な数字タイルの生成を目指すパズルゲーム「2048」を立体にし、回転や落下といった立体ならではの要素を加えたもの [13] がある。2D パズルゲームであるピクロスを立体化したもの [14] も存在する。これらと同様に、数独も立体化したバリエーションが存在する。これを立体数独と呼ぶ。立体数独は従来の数独に奥行の要素を追加したもので、仮に一辺のマスの数が 9 マスの場合、その盤面のマスの総数は 729 マスとなる。立体数独のアプリケーションとして RealSudoku3D[15] があり、さらに、田中ら [16] によって、このアプリのインターフェースを改善する研究が行われている。Web ページの「立体数独への扉」[17] では、この立体数独を含むいくつかの種類の数独の問題及び解説が掲載されている。しかし、この立体数独の分野においては、平面の数独のように研究が進んでいるとはいえない。先に述べたいくつかの数独の研究も、立体数独に適応できるとする手法等は存在していない。その原因として、マスの数が膨大であり、制約も複雑化していることから人力で問題を作成することが非常に困難であることが考えられる。ほかにも、盤面が立体であるため紙媒体で遊ぶことも困難であり、平面の数独のように誰もが気軽に遊ぶことができるわけでなく認知度も高くないことが考えられる。立体数独の問題の作成は、坪井 [18] が自動生成の研究を行っている。しかし、この研究で提案されている手法では、$4 \times 4 \times 4$ のサイズの問題の自動生成しか確認できていない。加えて、この研究の提案手法では $9 \times 9 \times 9$ のサイズの

問題の自動生成は処理時間から現実的ではない。そこで本研究では、問題の自動生成の高速化を図るため、バックトラックを用いた立体数独の問題を自動生成する手法を提案し、検証を行った。バックトラックとは、問題の解を求める際、不可能だと判明した時点で後戻りして別の解を探すアルゴリズムのことである。その結果、$4 \times 4 \times 4$ の問題の自動生成の高速化及び、$9 \times 9 \times 9$ の盤面の自動生成に成功した。この研究によって立体数独に関する知見を広げることにつながると考えられる。

1.2 本論文の構成

本論文では、第 2 章では数独のルールについて述べる。第 3 章では本研究における提案手法について述べる。第 4 章では提案手法の結果と考察を述べる。最後に第 5 章で本研究のまとめについて述べる。

第 2 章

立体数独のルールについて

本章では、立体数独のルールについて述べる。まず、一般的な数独のルールについて述べたのち、立体数独について説明する。

2.1　数独のルール

本節では数独のルールについて説明する。数独は、下の図 2.1 のような、9 × 9 のマスを 9 つの 3 × 3 のブロックに分けた盤面を用いる。この盤面に以下の条件を満たすように数字を当てはめていく。

1. すべてのマスに 1 から 9 いずれかの数字を当てはめる。
2. 横 1 列に同じ数字があってはいけない。
3. 縦 1 列に同じ数字があってはいけない。
4. ブロック内に同じ数字があってはいけない。

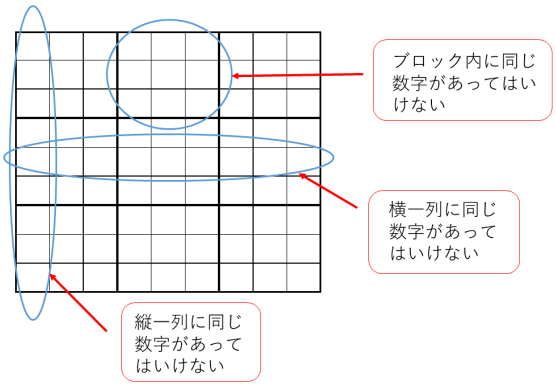

図 2.1　数独のルールに関する図

数独で遊ぶ際は、下の図 2.2 のようにヒントとなる数字があらかじめ盤面に存在する。このヒントから条件に基づき、空白のマスに当てはまる数字を推理し埋めていく。条件をすべて満たすことができれば完成である。

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

図 2.2　数独の問題盤面の例

2.2　立体数独のルール

本節では立体数独のルールについて説明する。立体数独とは下の図 2.3 のような、9 × 9 の盤面に奥行を追加し、9 × 9 × 9 の立方体にしたものである。この図は RealSudoku3D の実行画面である。全部で 729 マス存在し、これを以下のようなルールに基づき埋めていく。

1. すべてのマスに 1 から 9 のいずれかの数字を当てはめる。

2. 任意に選んだ 9 × 9 の盤面において、数独のルールを守る。

条件をすべて満たすことができれば完成である。

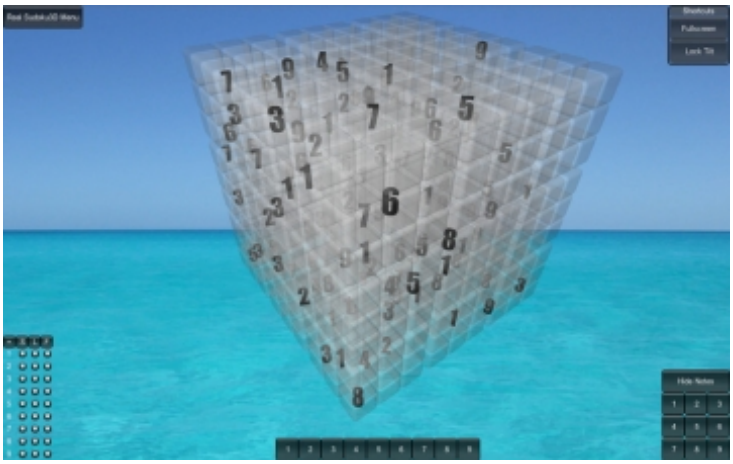

図 2.3　立体数独の盤面の図

2.3　数独における盤面のマスの数について

ここまで、数独の盤面は一辺が 9 マスの正方形であると述べてきた。しかし、数独のバリエーションとして、一辺のマスの数が 9 以外のものを作成することも可能である。数独における盤面のマスの数を変更する際には、ブロックのマスの数に注目する。数独の盤面において、ブロックの一辺のマスの数を n とすると、ブロック全体のマスの数は $n \times n$ マスとなり、盤面のマスの総数は $n^2 \times n^2$ マスとなる。さらに、ルール内の「すべてのマスに 1 から 9 のいずれかの数字を当てはめる」の部分を、「すべてのマスに 1 から n^2 のいずれかの数字を当てはめる」とする。これにより、この n に当てはめる数字によって盤面のマスの数を変えることができる。たとえば、n に 4 を当てはめると、下の図 2.4 のような一辺が 16 マスの盤面ができる。

			16				14	9		10	5		3		6
	4		2				1	15	13	16	8	11			
				12			11					15	5		
					15						6		13		10
7			15					5	12		4			10	3
	2														5
4	11	8				12				9	10		16		
	3	14	6	10	7					8		9		1	
		6	5			1		8				13			
15					12	16			9						
9	1		11		6	13								5	
12		7	8	11	5	15			14	6	16	10			9
							16				15	12		7	1
			14		2	6		4	8		1			13	
10	15		4				8			13			9	6	14
6		3				4			10	12				2	8

図 2.4　16 × 16 の数独

また、ブロック 2 × 3 のような長方形にすることで、下の図 2.5 のような、ブロックが正方形でない数独の盤面を作成することもできる。

1	3	4	5	2	6
2	5	6	1	4	3
5	1	2	6	3	4
6	4	3	2	5	1
3	6	5	4	1	2
4	2	1	3	6	5

図 2.5　6 × 6 の数独

立体数独においても、同様に一辺のマスの数を変更することが可能である。ブロックの一辺のマスの数を n とし、$n^2 \times n^2$ の盤面を奥行として n^2 個追加する。ルールに関しても同様の変更をおこなう。こうすることで、$n^2 \times n^2 \times n^2$ の立体数独を作成することが可能である。

第 3 章

提案手法

本章では、立体数独の問題を自動生成する手法について述べる。大まかな流れとして、初めに解答となる盤面を生成し、その後その盤面から空白のマスを増やし問題を生成するという手順を踏んだ。まず、解答となる盤面を作成する手法について述べたのち、問題の盤面を作成する手法について述べる。ただし、以下の説明はアルゴリズムの解説を目的とするため平面の数独を例とする。立体数独に対して行う場合は、盤面を立体化し、ルールを立体数独のものにすればよい。

3.1 バックトラックとは

数独は制約充足問題の一面も持つ。制約充足問題とは、変数の集合に対し、決められた領域の中から、与えられたいくつかの制約を同時に満たすような値を割り当てることを目的とする問題である。一辺が 9 マスの数独で例えると、空白のマスが変数、1 から 9 の数字が領域、第 2 章で述べる数独のルールが制約となる。そのほかの代表的なものに、$n \times n$ マスの盤面に n 個のクイーンを配置する N クイーン問題や、隣り合う領域を異なる色で塗りつぶす四色問題がある。このような問題に対して用いる代表的な手法の 1 つがバックトラック [19] である。バックトラックは深さ優先探索の一種であり、再帰手続きを用いて実現が可能である。

3.2　解答盤面の生成手法

本節では、立体数独の問題を作成していくうえで、最終的な解となる盤面の生成手法について述べる。本研究では、この解答盤面を生成する手法としてバックトラックを用いた。ここからバックトラックを用いた数独の問題盤面の作り方について、具体的な手順について説明する。

1. 初期盤面として、すべてのマスに 0 を埋めた盤面を用意する。

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

図 3.1　初期盤面

2. 探索する順番を指定する。今回は下の図 3.2 のようにする。これはあくまで探索順序を視覚化したものであり、マスに数字を埋めたわけではないことに注意する。

1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18
19	20	21	22	23	24	25	26	27
28	29	30	31	32	33	34	35	36
37	38	39	40	41	42	43	44	45
46	47	48	49	50	51	52	53	54
55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72
73	74	75	76	77	78	79	80	81

図 3.2　探索順序

3. 指定した順序に基づいて 1 から 9 の数字を数独のルールを破らないよう当てはめていく。

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

図 3.3　探索の途中経過

4. もし、現在探索しているマスに当てはめることのできる数字がない場合、1 つ前のマスに戻る。例として下の図 3.4 の赤く塗られているマスに注目する。このマスは、周辺の青く塗られている縦の列と横の列に注目すると、1 から 9 の数字がすべて存在しており、数字を当てはめることができない。よって、1 つ前のマスに戻りそこに別の数字を当てはめることを試す。

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	3	5	6	9	8	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

図 3.4　探索の失敗例

5. 0 が入ったマスがなくなれば完成である。

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

図 3.5　完成

この流れをフローチャートにしたものが下の図 3.6 である。

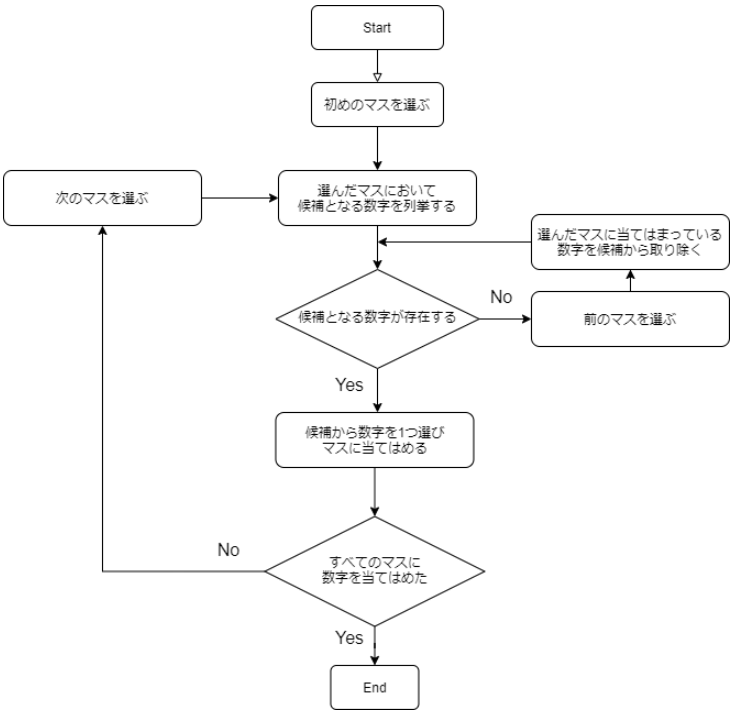

図 3.6　問題盤面の生成手法のフローチャート図

3.3　問題盤面の作成

本節では、問題盤面を作成する手法について述べる。まず、前節で作成した盤面からマスを 1 つ選び、以下の手順を行う。

1. そのマスの候補となる数字の数を調べる。
2. 候補数が 1 つであれば、そのマスには元から配置されている数字の代わりに 0 を配置する。下の図 3.7 の赤のマスは、青く塗られている縦の列、横の列、ブロックに注目すると、6 以外の数字を当てはめることができない。よってこのマスは 0 に書き換える。

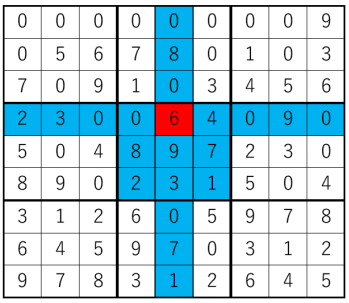

図 3.7　数字を 0 に書き換えられる場合

3. 候補数が 1 つでなければ、そのマスには元から配置されている数字を残す。下の図 3.8 の赤のマスは、青く塗られている縦の列、横の列、ブロックに注目するとすでに存在している 9 以外に 7 を当てはめることができる。したがってこのマスには 9 を残しておく。

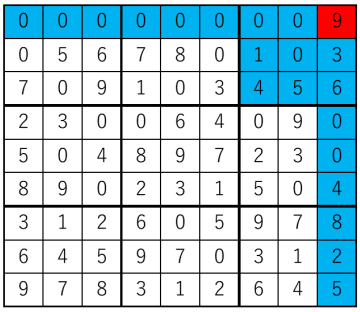

図 3.8　数字を書き換えられない場合

これを、すべてのマスで行えば完成である。ここで 0 が配置されているマスが、数独における空欄のマスとなる。この手法ならば、完成した盤面以外のすべての状態で候補が 1 つしか存在し

ないマスがあるため、問題の解が 1 つに定まることとなる。よって問題作成後に解が 1 つに定まるかどうかをチェックし、複数解が存在した際にやり直しをするといった処理を省くことができる。このことから時間の短縮を図ることができると考えたためこの手法を用いた。

第 4 章

提案手法の検証と考察

本章では、提案手法による実行結果および考察について述べる。今回は 4 × 4 × 4 の立体数独と、9 × 9 × 9 の立体数独の 2 つで問題の自動生成を試みた。制作環境は Visual Studio2019、ライブラリは FK[20] を使用した。検証環境は表 4.1 で示す。

表 4.1　検証環境

OS	Windows 10
CPU	Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz
メモリ	8.0GB SODIMM
GPU	Intel(R) HD Graphics 520

検証方法として、解答盤面の生成時間と問題盤面の生成時間の 2 つを計測するために、プログラムの実行時間を計測するコードを追加した。これを 4 × 4 × 4 と 9 × 9 × 9 の 2 つの盤面でそれぞれ 10 回ずつ行った。

4.1　4 × 4 × 4 での実行結果

提案手法による 4 × 4 × 4 の盤面の生成結果は以下の図 4.1 から図 4.10 のようになった。

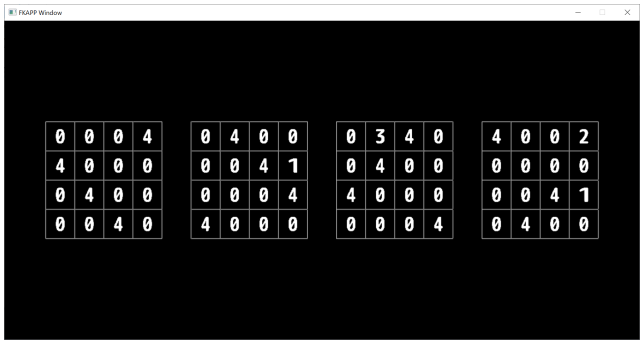

図 4.1　実験結果 01

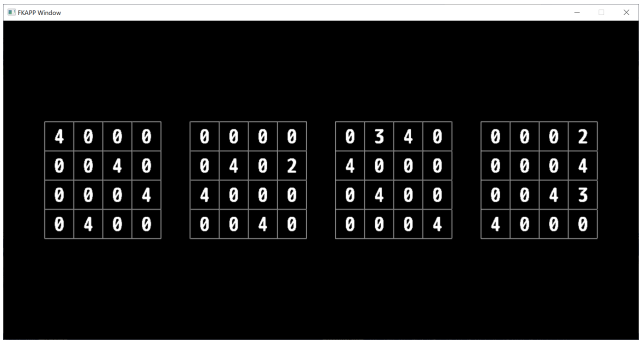

図 4.2　実験結果 02

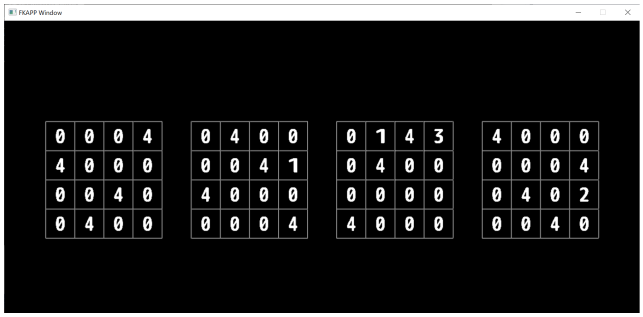

図 4.3　実験結果 03

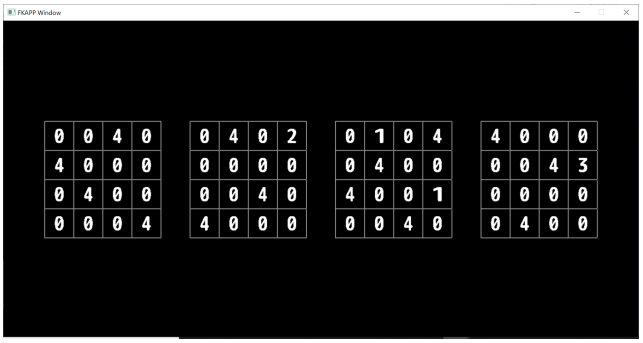

図 4.4　実験結果 04

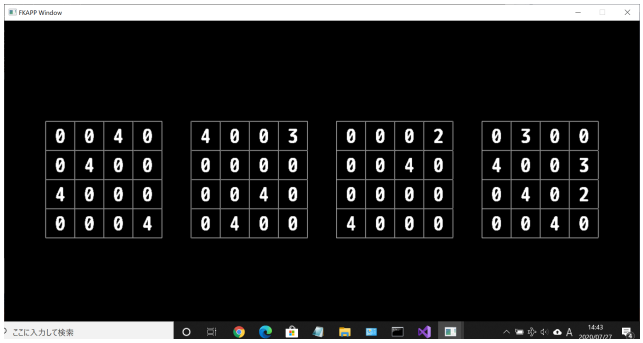

図 4.5　実験結果 05

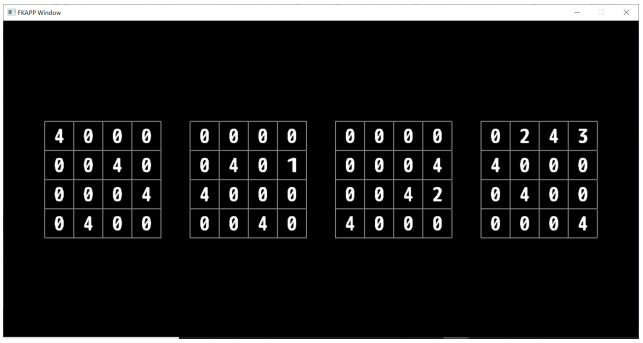

図 4.6　実験結果 06

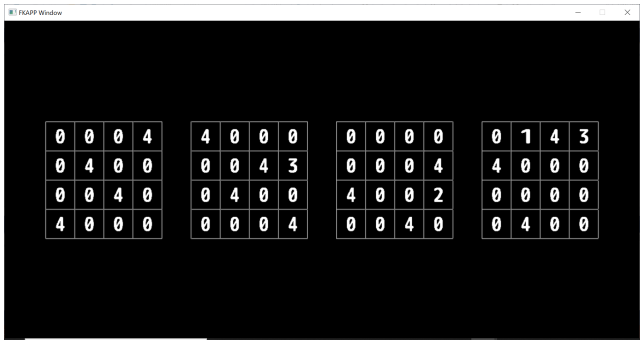

図 4.7　実験結果 07

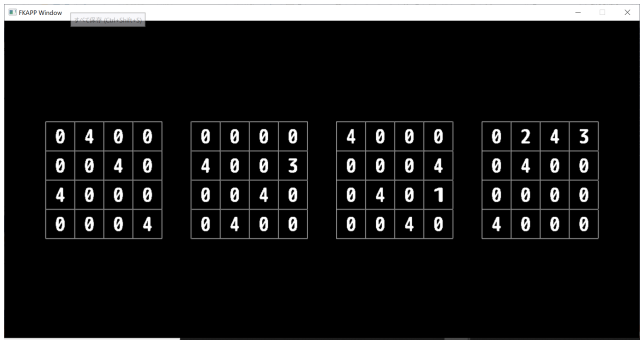

図 4.8　実験結果 08

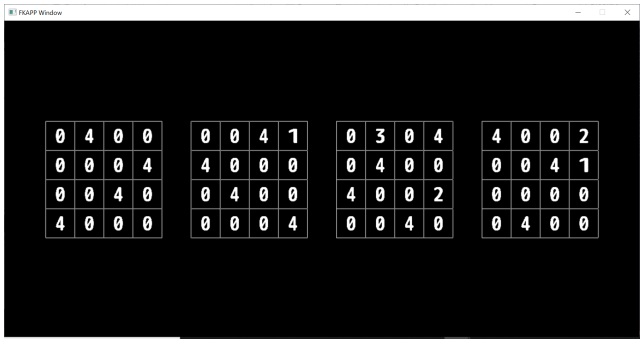

図 4.9　実験結果 09

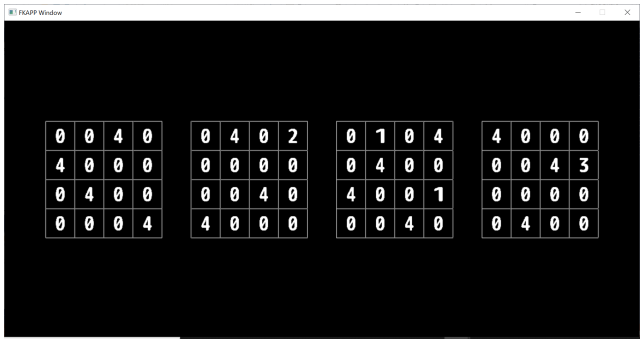

図 4.10　実験結果 10

下の表 4.2 は、図 4.1 から図 4.10 の生成時間をまとめたものである。

表 4.2　4 × 4 × 4 の問題盤面生成時間に関する検証結果

		解答盤面	問題盤面	合計値	図番号
生成時間 (単位:秒)	1 回目	0.248	0.003	0.251	4.1
	2 回目	0.239	0	0.239	4.2
	3 回目	0.311	0.001	0.312	4.3
	4 回目	0.234	0.002	0.236	4.4
	5 回目	0.219	0.003	0.222	4.5
	6 回目	0.231	0.017	0.248	4.6
	7 回目	0.240	0	0.240	4.7
	8 回目	0.282	0.002	0.284	4.8
	9 回目	0.262	0.002	0.264	4.9
	10 回目	0.226	0	0.226	4.10
	平均値	0.2492	0.003	0.2522	-

上の表で問題盤面の生成時間が 0 秒となっている場合があるが、これは処理速度の速さから測定不能であったため 0 秒として扱っている。

4.2　9 × 9 × 9 での実行結果

提案手法による 9 × 9 × 9 の盤面の自動生成は、以下の図 4.11 から 4.14 のようになった。ただし、10 回の実行において膨大な時間がかかってしまったため、実行時間が 4 時間を超えて生成できなかったものは取り扱わなかった。その結果、生成できたものは 4 つ、できなかったものは 6 つとなった。

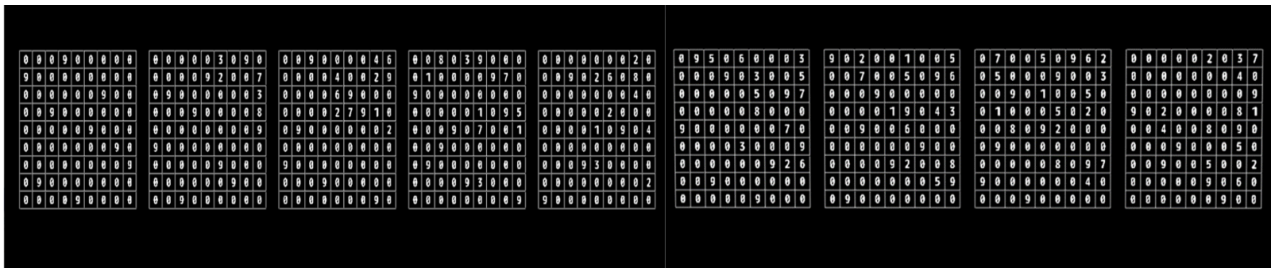

図 4.11　実験結果 03

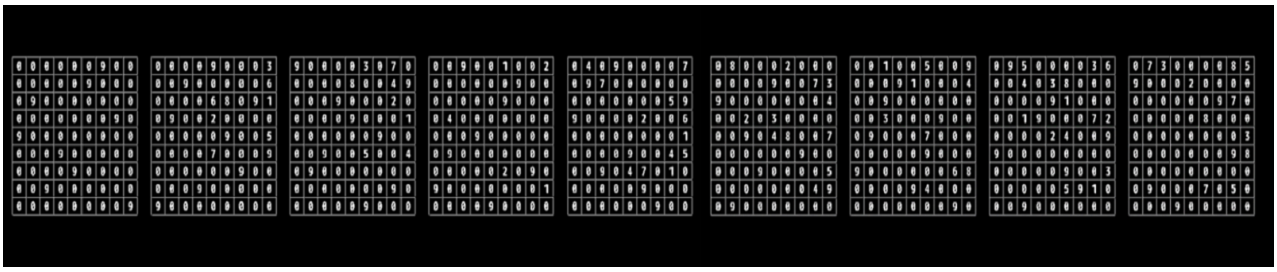

図 4.12　実験結果 04

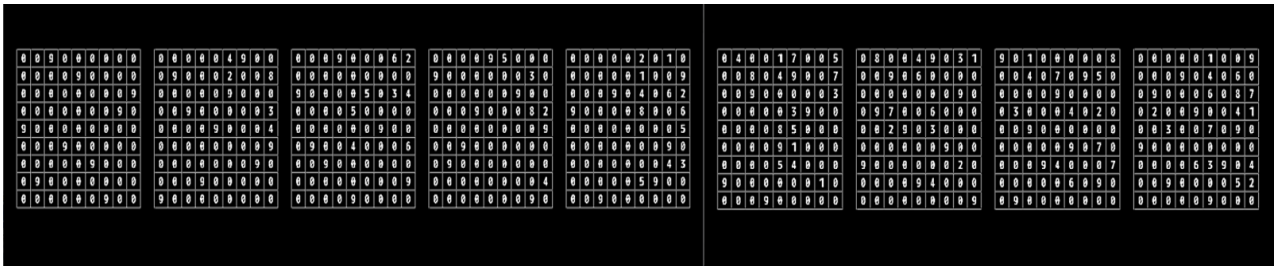

図 4.13　実験結果 06

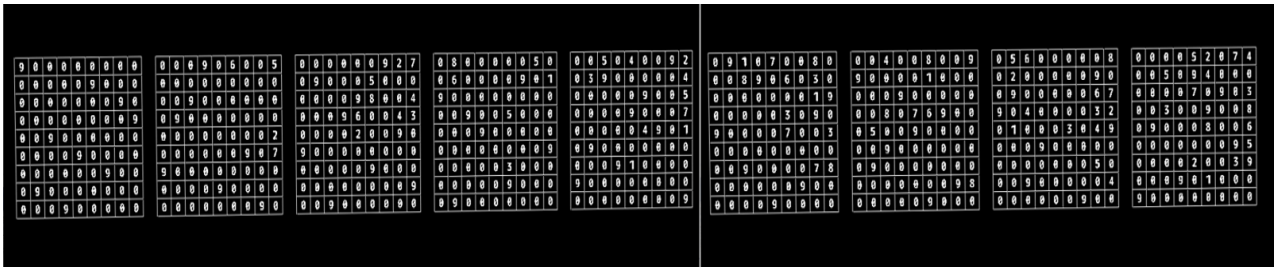

図 4.14　実験結果 07

詳細な結果は下の表 4.3 のとおりである。また、ここでの平均値は、問題盤面の作成に成功した値のみをを用いて算出した。

表 4.3　9 × 9 × 9 の問題盤面生成時間に関する検証結果

		解答盤面	問題盤面	図番号
生成時間	1 回目	4 時間以上未完成		-
	2 回目	4 時間以上未完成		-
	3 回目	27 分 23 秒	0.031 秒	4.11
	4 回目	2 時間 2 分 59 秒	0.031 秒	4.12
	5 回目	4 時間以上未完成		-
	6 回目	1 時間 34 分 21 秒	0.02 秒	4.6
	7 回目	3 時間 58 分 23 秒	0.02 秒	4.7
	8 回目	4 時間以上未完成		-
	9 回目	4 時間以上未完成		-
	10 回目	4 時間以上未完成		-
	平均値	2 時間 0 分 46 秒 5	0.025 秒	-

4.3　考察

提案した手法によって 4 × 4 × 4 の立体数独の自動生成に成功した。参考として、坪井の研究の結果を見てみると、生成時間は平均して約 69 秒となっている。本研究の結果である平均約 0.25 秒と比べると、提案手法を用いることで 276 倍の高速化に成功している。実験環境は異なっているため正確な数値ではないが、コンピュータの性能に大きな差はないため誤差はそれほど大きく

ないと考えられる。参考に、下の表 4.4 は坪井の研究で用いた PC のスペックである。

表 4.4　坪井の研究における検証環境

OS	Windows 8.1(64bit)
CPU	Intel Core i7-4790 3.60GHz
メモリ	8.00GB 2DIMM
GPU	NVIDIA GeForce GTX 970

$9 \times 9 \times 9$ の盤面においても自動生成に成功した。しかし、膨大な時間がかかっているうえに、それでも完成しない場合もあり、確実に自動生成が可能であるとは言えない結果となった。これは提案手法においてあらかじめ探索する順番を決めたことが原因と考える。数独の制約によって、数字を当てはめたマスから割り当てた数字の遠いマスの候補が存在しなくなってしまうことがある。今回の提案手法ではこれを調査しないため、無駄な探索が非常に多くなってしまっている。この候補を考慮することで、より高速に問題の生成が可能になると考える。

第 5 章

まとめ

本研究では、立体数独における問題の自動生成を行った。手法として、バックトラックを用いて $4 \times 4 \times 4$ の盤面と $9 \times 9 \times 9$ の盤面の 2 種類において自動生成を試みた。結果として、$4 \times 4 \times 4$ の盤面の立体数独の自動生成は短時間で行うことができるが、$9 \times 9 \times 9$ の自動生成は膨大な時間がかかることが分かった。これは、今回の提案手法では探索の効率が悪く、場合によっては大量にバックトラックにて後戻りをしてしまったためだと考えられる。そのため、今後の課題として、$9 \times 9 \times 9$ の盤面において、より高速に自動生成するために探索を効率化するアルゴリズムを考案する必要があると考えられる。

謝辞

本研究を進めるにあたり、ご指導いただいた渡辺先生、阿部先生に深く感謝いたします。また、論文執筆の際多大な迷惑をかけながらも応援してくれた家族、友人たちに感謝いたします。

参考文献

[1] 遥小河, 智明井本, 伸明武藤. 数独エントロピーを用いた数独ソルバーの高速化. 第 80 回全国大会講演論文集, 第 2018 巻, pp. 57–58, mar 2018.

[2] 一貴前田, 博奥乃. 数独の問題作成支援システムの設計と開発. 全国大会講演論文集, 第 70 回コンピュータと人間社会, pp. 799–800, mar 2008.

[3] 小場隆行, 中所武司. 数独の難易度判定アプリケーションの提案と評価. Technical Report 8, 明治大学大学院ソフトウェア工学研究室, 明治大学大学院ソフトウェア工学研究室, feb 2011.

[4] Gary McGuire, Bastian Tugemann, and Gilles Civario. There is no 16-clue sudoku: Solving the sudoku minimum number of clues problem, 2012.

[5] 翔座間, 功篠埜. 初期配置が指定された場合に適した数独問題生成手法の提案および実装. Technical Report 1, 芝浦工業大学, 芝浦工業大学, mar 2016.

[6] 律政那須, 公紀松崎. モンテカルロ木探索による数独少数ヒント盤面の生成. 第 54 回プログラミング・シンポジウム予稿集, 第 2013 巻, pp. 173–180, jan 2013.

[7] 湧古川, 修身山本. 確率的逐次添加法によるヒントの少ない数独問題の生成. 第 82 回全国大会講演論文集, 第 2020 巻, pp. 91–92, feb 2020.

[8] T. Mantere and J. Koljonen. Solving, rating and generating sudoku puzzles with ga. In *2007 IEEE Congress on Evolutionary Computation*, pp. 1382–1389, 2007.

[9] Amit Benbassat. Genetic algorithms are very good solved sudoku generators. *GECCO '19: Proceedings of the Genetic and Evolutionary Computation Conference Companion*, p. 49–50, 2019.

[10] 康夫松原. 数独の推論規則と難易度に関する考察. Technical Report 134(2006-EC-005), 文教大学情報学部情報システム学科, dec 2006.

[11] 哲広大谷, 聖松田. ニューラルネットワークによるパズルの求解 – ホップフィールドネットワークで数独は解ける –. Technical Report 16(2009-ICS-154), 日本大学大学院生産工学研究科, 日本大学生産工学部, feb 2009.

[12] 山村明弘藤原美早紀. 立方体上の n-クイーン問題と n-ルーク問題. 情報処理学会論文誌, Vol. 53, pp. 1592–1601, 2012.

[13] オリジナル 3D Puzzle Game【SPIN & DROP】iOS 版アプリリリース！ `https://alterbo.jp/blog/news12/`. 参照:2020:08:03.

[14] 立体ピクロス. `https://www.nintendo.co.jp/ds/c6pi/index.html`. 参照:2020.07.027.

[15] RealSudoku3D. `http://www.realsudoku3d.com/`. 参照:2020:08:03.

[16] 白石路雄田中貴拓. 立体数独アプリケーションの開発. 映像メディア学会技術報告, Vol. 39, pp. 187–190, 2015.

[17] 立体数独への扉. `http://w01.tp1.jp/~sr10026691/MainSJ.html`. 参照:2020.09.15.

[18] 立体数独の自動生成に関する研究. `https://gamescience.jp/2017/Paper/Tsuboi_2017.pdf`. 参照:2020:08:03.

[19] 伊庭斉志. ゲーム AI と深層学習. 株式会社オーム社, 2018 年.

[20] Fine Kernel ToolKit System. `https://gamescience.jp/FK/`. 参照:2020.07.027.