2017 年度　卒　業　論　文

立体数独の問題の自動生成に関する研究

指導教員：渡辺 大地 准教授

メディア学部 ゲームサイエンス プロジェクト
学籍番号　M0113306
坪井　健

2018 年 3 月

2017 年度　　卒　業　論　文　概　要

論文題目

立体数独の問題の自動生成に関する研究

メディア学部 学籍番号：**M0113306**	氏 名	坪井　健	指導 教員	渡辺 大地 准教授

キーワード	パズル、立体数独、FisherYates shuffle、自動生成、解法

世界には数多くの、数字を用いたパズルゲームが存在し、2048 や数蛇、ラテン方陣などが例として挙げられる。その代表格として、数独というパズルゲームがある。これは、3 × 3 の範囲をもったブロックに区切られたマスに、1 から 9 の数字を入れていき、すべてのマスを埋めることを目的としたパズルゲームである。数独は昔から様々な国で愛されており、世界選手権が開催されるほどである。バリエーションの多さも数独の魅力の一つである。1 ブロックあたりの範囲や用いる数を変更したもの、ブロックの形状を変化させたもの、隣り合ったマスの数字によってそのマスの数字が限定されるものなどが挙げられる。数学的な観点からも、数独は非常に興味深いものとして取り上げている。解が 1 通りになる中で、ヒントの数はいくつまで開示してよいのか、解のパターン数は何通りあるのかといった研究や、問題を自動で生成するための研究もおこなわれている。その数独の新たなバリエーションとして、立体数独というものが存在する。これは、一般的な数独に奥行きの概念を加えた、正六面体の形状をしたパズルである。埋めなければならないマスの数は一気に増え、奥行きも考えて数字をあてはめなければならないため、問題としての難易度は一気に上昇する。また、展開図を考えながら問題を把握しなければならないため、全体像が把握しにくいといった問題もある。そうした事情も含め、立体数独は研究が進んでいない分野である。本研究では、いまだ行われていない,4 × 4 × 4 の立体数独の問題を自動生成するプログラムの開発を行い、数独の規則に沿った、解が 1 通りになる問題を生成できたかの検証を行った。その結果、現在のプログラムで、4 × 4 × 4 の立体数独の問題を正しく生成できることが確認できた。

目 次

図 目 次

第 1 章

はじめに

世界には数字を用いたパズルゲームが多く存在する。表示された数字を足していき、2048 になるよう選択していくパズルゲーム [1] や、四則演算を用いて、表示された数字を組み合わせ、10 になるような計算式を作るパズルゲーム [2] も存在する。これらの数字を用いたパズルゲームの代表格の一つとして、数独というパズルゲームが存在する。数独は、ラテン方陣と同様の n 行 n 列と n 種類の連続した要素からなるが、ラテン方陣とは違い、ボックスと呼ばれる正方形の仕切りが存在し、同一のボックス内では同じ要素を入れることができないというルールが追加される。また、数独の全体図はボックスの集合体ともみることができ、$n = 2^2, 3^2, 4^2...$ となる。数独の名称は、「数字は独身に限る」という正式名称の略称である。また、「数独」と「ナンプレ」の 2 つの名称を持っており、パズル制作会社ニコリが製作に関わっている場合は「数独」、関わっていない場合は「ナンプレ」となる。世界選手権も開かれるほどであり、その問題の多様性とルールの単純さから、幅広い人気を誇っている。Weblio 辞書 [3] によると、現在の数独の始まりは、18 世紀にスイスの数学者レオンハルト・オイラーが考案したラテン方陣であり、これにアメリカの建築家ハワード・ガンスが 3 × 3 のボックスとそれに関するルールを付け加え、ペンシルパズルとしたものが現在の数独の形である。当時は「ナンバープレース」という名称で出版された。日本

では、株式会社ニコリから 1984 年に初めて紹介され、1988 年に単行本が出版された。このころからニコリによる公式名称として「数独」が用いられ始めた。「数独」という名称が一般的になるのは 1992 年からである。数独はその性質上、解となりえる盤面の通りが非常に膨大に存在する。Russell ら [4] は、回転や反射や順列や名前を変更することなどの左右対称を考慮に入れた場合ですら、54 億 7273 万 538 通りも存在することを示した。また、数独には様々なバリエーションも存在する。バリエーションの種類は数多く存在しており、対角線上の数字も 1 から 9 を各一回づつ使うルールを追加したものや、本来 3 × 3 のボックス型の仕切りを変形させたものも存在する。また、パズルとしての数独という側面のほかに、数学的な研究対象としての数独という側面も存在する。2007 年に、藤原 [5] が 1 問を 0.05 秒以下で作成する性能を持った、問題を自動生成するプログラムを開発した。このシステムの特徴として、パズル作家の考え方の特徴と取り入れることで、良問とされる問題を短時間で生成できる点が挙げられる。このシステムが生まれた背景には、現在の粗製乱造されている数独問題が、良問と悪問の区別をつけられていないという問題がある。参考文献中で良問と悪問の定義は、解くことで脳が悪化する問題や解くことがつまらない問題が悪問とされている。このプログラムのほかにも、そのマスに入る数値のあるなしで入る可能性のある数値を決定することで、問題を作成する際の補助になるシステムの考案も、前田ら [6] によって行われた。この論文での研究結果として、ヒント数が 24 以下になると問題の生成が急激に困難になるという研究結果がある。ヒント数に関しての研究はほかにもあり、那須ら [7] は、9 × 9 の問題生成の場合では、ヒント数が 20 以下の問題は作成することができず、生成できた問題でも、複数回を持つ場合の問題を生成する確率が高い傾向が見られたことを発見した。これが問題の自動生成の手法に関連する問題なのか、数独の性質からくる問題なのかは、参考文献中では明らかにはならなかった。また、モンテカルロ木探索を用いた、唯一解をもつ問題を構成するために必要なヒントの、最小数を探る研究も那須ら [**?**] によって行われており、その中で示されているヒント数が最も少ないものは 17 であり、1 問のみの発見となった。そして、小場ら [8] によっ

て数独問題の難易度をアプリケーションによって判定することで、評定者によってまちまちになりがちな難易度評価を規格化しようとする研究を、棟方ら [9] は、手のひらの電気活動を計測し、回答者を基準とする難易度推定をしようとした研究を行った。井上ら [10] による、本質的に異なる解の盤面を列挙し、番号付けを行った研究もある。この研究により、5472730538 番目までの番号付けが完了した。谷尾里ら [11] によって、デザインを重視した問題を生成するために、ユーザーが入力した画像をもとにヒントマスの位置を自動で設定できるようにした、数独の問題を自動生成するシステムの提案も行われた。また、二分決定グラフを用いて、数独パズルの解探索と列挙を行った研究も、立石ら [12] によって行われた。数独を解く際に用いることができる推論規則はどういったものがあるのかを考察した研究も松尾 [13] によって行われた。このように、数独は、問題をどうやって解くか、問題をどうやって作るか、解はいくつあるのかなどを、様々な手法を用いて研究されている分野でもある。しかし、これらの研究は、平面である通常の数独において適応できる提案や手法であり、少なくとも、立体的な数独問題に適応できるとしている提案や手法は現在存在していない。そのような数独の 3 次元版ともいうべき、立体数独というバリエーションが存在する。これは、行と列しかなかった数独に奥行きの概念を加え、より複雑化させた数独問題のバリエーションの一つである。これによって、埋めるべきマスの総数は、一般的な立体数独の形状である 9 × 9 × 9 の場合、ヒントとなる要素を含めて、9 × 9 × 9=729 マスにもなり、非常に解きごたえのある問題となっている。立体数独の研究としては、立体数独の問題を解答するためのアプリケーションには RealSudoku3D というソフトの存在を踏まえた、立体数独をより直感的にとらえるために、Unity を用いて問題を立体表記する研究が、田中ら [14] によって行われた。以下の図 1.1 は RealSudoku3D の実行画面、図 1.2 は田中らによって開発された研究成果の実行画面である。

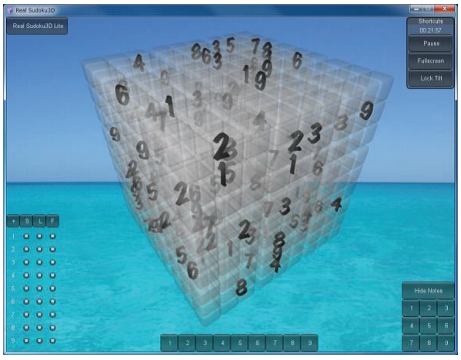

図 1.1　RealSudoku3D の実行画面

研究結果では、RealSudoku3D よりも操作性がよく、問題を解いた際の達成感という点でも有意差が得られたという実験結果を得ている。

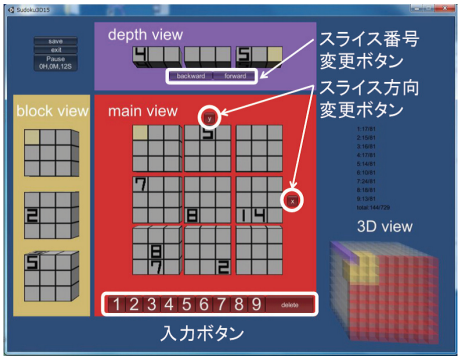

図 1.2　Unity を用いて問題を立体表記する研究の実行画面

しかしながら、ここまで数独が複雑化してしまうと、人力で問題を生成することが非常に困難である。また、立体数独という存在が一般的にはほとんど認知されていないからか、立体数独の問題を自動生成するソフトウェアは存在していない。よって本研究では、立体数独における問題の自動生成を行うソフトウェアの開発を行った。この研究によって、問題の作成が難しい立体数独の問題を、自動生成できるようになり、それにより立体数独に対する知見を広めることにつながると考えられる。それと同時に、新しい立体数独の問題に気軽に挑戦できる環境を構築することができる。そこで本研究では、有限の配列の順序をシャッフルする Fisher Yates Shuffle[15][16] に着

目し、あらかじめ行単位で 1 から 4 を順に繰り返し入れた盤面を、行の中で Fisher Yates Shuffle を用いて、数独のルールが守られた盤面が生成されるまで繰り返す手法で数独の問題を生成した。この手法で盤面を 4 つ生成し、立体数独の盤面間のルールに沿う形になるまで、各盤面の生成を繰り返すという手法で立体数独の問題を生成した。最後に、正しく問題が生成されているか検証を行った。その結果、本研究の目的である 4 × 4 × 4 の立体数独の問題の自動生成に成功した。

1.1 本論文の構成

本論文では、2 章では数独と立体数独の概要について述べる。3 章では提案手法の解説について述べる。4 章では提案手法の検証と結果について述べる。5 章で考察についてを述べ、6 章でまとめを述べる。

第 2 章

数独とは

本章では、数独のルールと解法の種類、問題を作成する際の手法、立体数独のルールについて説明する。

2.1　数独のルール

数独は、問題作成時に既に埋められているヒントマスをもとにして、空いているマスに要素を入れていく [17]。その際に、以下の 4 つのルールを守らなければならない。これらのルールは、数独のバリエーションの違いによって詳細な部分が変化する場合がある。

1、n 行 n 列すべてのマスを、1 から n の要素を用いて埋める

2、同列で同じ数字を使わない

3、同行で同じ数字を使わない

4、同ボックス内で同じ数字を使わない

これらのルールを守りながら数字を空いているマスに埋めていき、すべてのマスで数字を入れることがでできれば、数独パズルの完成となる。要素として主に使われるのは数字であり、例えば 4 × 4 の数独問題の場合、マスの合計は 16 マス、使用する要素は 1 から 4 となる。バリエー

ションによっては、A から I のアルファベットであったり、ボックスの形状が正方形でない場合もある。

2.2　数独の解法の種類と生成に際しての注意点

数独の問題を生成するためには、回答に空きマスを作るという方法が一般的である。また、その問題の解が 1 通りになるよう注意を払わねばならない。よって、数独の解き方を学ぶことで、空きマスを設けた問題が、解が 1 通りになるかどうかの確認ができる。この節では、数独の解法の種類について解説する。数独の空きマスを埋める際に活用される手法は数多くあり、基本ルールからなる基本的な手法から入る数字の候補を利用した、中級、上級手法が存在する。中にはこれらの手法を組み合わせないと解けないような複雑な問題も数多く存在する。ここでは解法を紹介しているサイト [18][19] や教本 [20] から、数独を解くために必要な手法を紹介する。手法解説のため、数独のマス目を、行を x_1 から x_9、列を y_1 から y_9 にそれぞれ割り当てる。例えば、(x_4, y_6) と表記された場合、以下の図 2.1 の x_4 と y_6 の交差する場所を表している。

	x1	x2	x3	x4	x5	x6	x7	x8	x9
y1									
y2									
y3									
y4									
y5									
y6				(4,6)					
y7									
y8									
y9									

図 2.1　手法解説での (x_4, y_6) の位置

そして、$\{(x_i, y_j) \mid 1 \leq i \leq 3, 1 \leq j \leq 3\}$ の範囲をボックス 1、以下同様に、$4 \leq i \leq 6, 1 \leq j \leq 3$ の範囲をボックス 2、$7 \leq i \leq 9, 1 \leq j \leq 3$ の範囲をボックス 3、$1 \leq i \leq 3, 4 \leq j \leq 6$ の範囲をボックス 4、$4 \leq i \leq 6, 4 \leq j \leq 6$ の範囲をボックス 5、$7 \leq i \leq 9, 4 \leq j \leq 6$ の範囲をボックス 6、$1 \leq i \leq 3, 7 \leq j \leq 9$ の範囲をボックス 7、$4 \leq i \leq 6, 7 \leq j \leq 9$ の範囲をボックス 8、$7 \leq i \leq 9, 7 \leq j \leq 9$ の範囲をボックス 9 とする。数独を解く基本的な手法として、列と行、そして同ボックスに同じ数字が入らないというルールを利用する手法がある。例えば、ボックス 4 の (x_3, y_4) を除くすべてのマスに、1、2、4，5，6，7，8，9 が入っている場合、(x_3, y_4) には 3 以外の数字がルール上はいらないため、このマスの数字は 3 と確定できる。同様に、(x_1, y_5) から (x_8, y_5) の同行上に、1、2、4，5，6，7，8，9 が入ってい場合、同列行に同じ数字が入らないというルールから、(x_9, y_5) には 3 が入ると確定できる。以下の図は、(x_3, y_4) と (x_9, y_5) のマスが確定している状況の図 2.2 である。

	x1	x2	x3	x4	x5	x6	x7	x8	x9
y1									
y2									
y3									
y4	1	2							
y5	4	5	6	1	2	7	8	9	
y6	7	8	9						
y7									
y8									
y9									

図 2.2　(x_3, y_4) と (x_9, y_5) のマスに 3 が入ることが確定している

また、ボックス 1、ボックス 5、ボックス 6、ボックス 7 に入る 3 の位置が $(x_1, y_8)(x_3, y_2)(x_6, y_5)(x_9, y_6)$ と判明している場合、3 が入っているマスの位置と、同行列に同じ数字が入らないというルールから、$(x_1, y_4)(x_3, y_4)(x_1, y_5)(x_2, y_5)(x_3, y_5)(x_1, y_6)(x_2, y_6)(x_3, y_6)$ には 3 が入らないため、(x_2, y_4) に 3 が入ることが確定する。この手法を単独候補マスという。同

様に、(x_3, y_9) のマスを見たときに、同ボックス内で 1、3、4、5、6、x_3 列で 8、9、y_9 行で 2 が使われている場合、同じ列、行、ボックス内で同じ数字を使わないというルールから、(x_3, y_9) には 7 が入ることが確定する。以下の図 2.3 は、(x_2, y_4) と (x_3, y_9) において、単独候補マスが成立している状態を表している。

	x1	x2	x3	x4	x5	x6	x7	x8	x9
y1			9						
y2			3						
y3			8						
y4	×		×						
y5	×	×	×			3			
y6	×	×	×						3
y7	1	2	4						
y8	3	5	6						
y9									

図 2.3　ボックス 7 において 3 と 7 の位置が確定している

これら以外にも、Naked Pair や三国同盟、独立マトリックスなど、中級以上の様々な解法が存在するが、本研究では取り扱わないので省略する。前述した通り、数独の問題を作成する際には、解が 1 通りになるように空白マスを作成する必要がある。解が 1 通りにならない問題であれば、解いている途中に問題の解が変わってしまい、解として成立していないことになる。それを防ぐために、就活分析録 [21] で紹介されているような、解を求める手法を用いた、解が 1 通りになるかどうかのチェック機構をシステムに組み込む必要がある。本研究では、問題の成立を目的とするため、中級以降の手法を用いなければならない状態については、問題の成立はできなかったこととする。よって、基本的なルールに基づいた手法のみを取り扱う。

2.3　立体数独のルール

立体数独のルールは、通常の数独を解く際の数字を確定させる要素に、奥行きにある数字の情報が追加される。よって、とあるマスに対して、列と行と奥行きの 3 方向をみて、同一数字が入らない数字を選択する必要がある。また、X 方向、Y 方向、Z 方向の 3 方向から立体数独を見たとき、それぞれの方向から見た際の同行列、同ボックス内で同じ数字を用いてはならない。

第 3 章

提案手法

この章では、立体数独の問題を自動生成する手法について述べる。本研究では、4 × 4 × 4 の問題生成の研究と生成を行った。4 × 4 × 4 の立体数独は 4 × 4 の数独問題を奥行き方向に積み重ねた形をしている。そのため、まずは 4 × 4 の数独問題を複数生成することに取り掛かり、それを並べることで 4 × 4 × 4 を生成した。4 × 4 の数独は、9 × 9 の数独のバリエーションの一つであり、用いる数字が 1 から 4 であること、盤面が 4 × 4 の 16 マスであること、ボックスの大きさが 2 × 2 であることが一般的である 9 × 9 の数独と異なる点である。一般的な 9 × 9 × 9 の立体数独ではなく、4 × 4 × 4 の立体数独を取り上げたのは、検証実験を行う際、9 × 9 × 9 の立体数独では必要となる計算量が非常に膨大になると予測し、実用的な計算時間に収まると思われる 4 × 4 × 4 の立体数独の方が適していると考えた。また立体数独を娯楽としての観点から考えた際、9 × 9 × 9 の立体数独の問題は解答するのが非常に難しく、一般的とは言えないという理由も存在する。

3.1 Fisher Yates Shuffle とは

Fisher Yates shuffle[15][16] とは、有限な長さの集合から、ランダムな順列を作るためのアルゴリズムである。例えば、1 から 9 までの数字が順に入った配列がある場合、最初の配列の並びは 123456789 の順である。これを Fisher YatesShuffle でシャッフルすると、123856759 や 823456719 になる。具体的な処理手順は以下の通りである。

1、入れ替えたい配列と、一時的に数字を入れる変数 a を 1 つ用意する。

2、配列の中で入れ替える先をランダムに決定する。

3、ランダム先の数字を変数 a に入れる。

4、配列の 1 番目の数字を、ランダムで決めた先の数字に入れる。

5、変数 a の値の値を配列の 1 番目に入れる。

これを 2 番目の数字と入れ替える、3 番目の数字と入れ替える・・・というように繰り返して、全体をシャッフルする。最終的な結果は 467285139 や 925813476 のようになる。このアルゴリズムを用いて、あらかじめ同行内で左端のマスから右端のマスにかけて 1 から 4 が順に入っている各盤面の各行のマスごとに、同行列とボックス内で同じ数字が用いられないような盤面になるまでシャッフルし、4 × 4 の数独の問題生成を行った。

3.2 4 × 4 × 4 の数独の生成手法

4 × 4 × 4 の立体数独の問題を生成するために、まず 4 × 4 の数独の解答の盤面 1 の生成を行った。次に、生成した盤面 1 と奥行き方向の数字のかぶりが起こらないような 4 × 4 の盤面 2 を生成した。同様の手順で盤面 3、盤面 4 を生成し、奥行き方向に並べるといった手順で、4 × 4 × 4 の立体数独の問題を生成した。

3.3　4 × 4 の数独の解答の自動生成

まず、すべてのマスに、左下から右上にかけて、左から右へ順に (0,0) から (3,3) の座標を割り当てた。次に、1 から 4 の数字を (0,0) から (3,0)、(0,1) から (3,1)、(0,2) から (3,2)、(0,3) から (3,3) にかけて順に入れたものを用意した。具体例としては、(0,0) には 0 を (0,3) には 4 を、(2,3) には 3 を入れた。それを行ごとで、Fisher Yates Shuffle アルゴルズムを用いてシャッフルした。その後、出来上がった 4 × 4 の盤面が数独のルールである、同行と同列、2 × 2 マスで区切られたボックス内に同じ数字が使われていないか確認をし、確認時点でルールに沿っていなければ、再度各行ごとにシャッフルを行い、盤面を生成する。確認時点でルールに沿っていれば、4 × 4 の盤面が完成する。4 × 4 の盤面の生成手順の全体の流れをフローチャートにしたものが図 3.1 である。

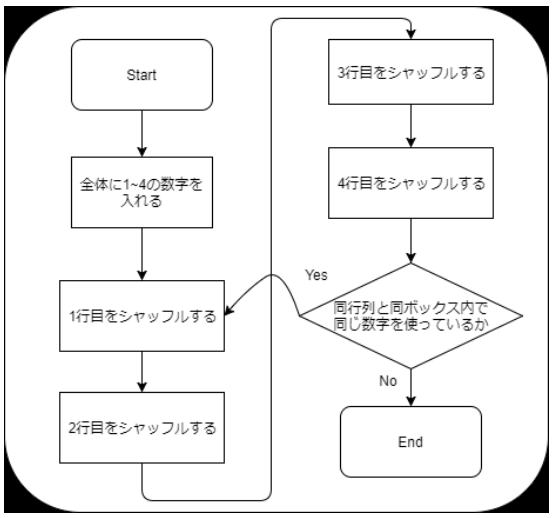

図 3.1　4 × 4 の数独問題の解答の盤面を生成するフローチャート図

以下の図 3.2 は、完成した盤面の例である。

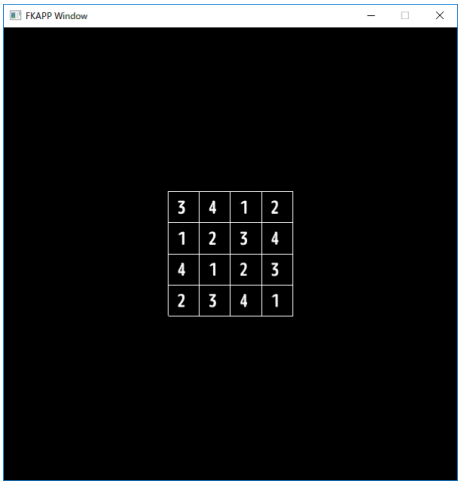

図 3.2　完成した 4 × 4 の数独問題の解答

各行列で同じ数字がかぶっておらず、また同ボックス内で、同じ数字が使われていないことが確認できる。

3.4　4 × 4 × 4 の数独の解答の自動生成

前節で生成した 4 × 4 の数独の解答の盤面 1 と奥行き方向で数字がかぶらないような、4 × 4 の数独の解答の盤面 2 を生成する。次に、盤面 1 と 2 とも奥行き方向で数字がかぶらないような盤面 3 を生成した。そして、それを横並びにしたのが図 3.3 である。

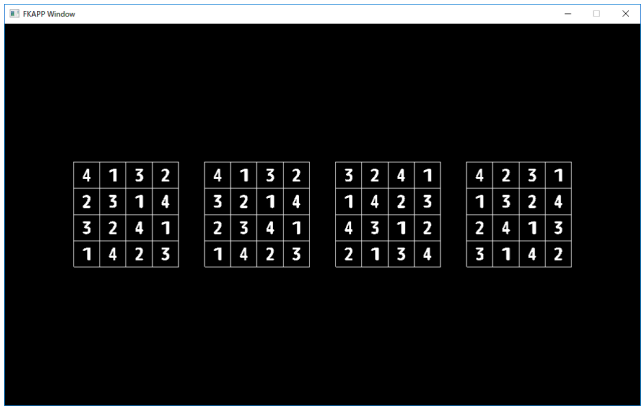

図 3.3　4 × 4 の数独問題の回答を 4 つ横並びにした結果

この状態のままでは、奥行き方向で数独のルールを満たしていないので、奥行き方向でも数独のルールが守られる並びになるまで、盤面の再生成を繰り返す。具体的には、2 番目に生成した盤が、最初に生成したの盤と、同奥行き内で数字がかぶっていた場合、2 番目に生成した番をもう一度各行ごとにシャッフルを行う。これを、2 番目に生成した盤の数字が、最初に生成した盤の数字と、奥行き方向で被らないようになり、かつ同行列と 2 × 2 のボックス内部で同じ数字が入らないようになるまで繰り返す。3 番目の盤を生成は、1 番目に生成した盤と 2 番目に生成した盤の両方と、奥行き方向で同じ数字が被らないようになり、かつ盤面内で同行列と 2 × 2 のボックス内部で同じ数字が入らないようになるまで、3 番目の盤の各行ごとのシャッフルを繰り返す。4 番目の盤の生成も同様に、1 番目と 2 番目と 3 番目の盤の数字と奥行き方向で同じ数字が被らないようになり、かつ同行列と同ボックス内で同じ数字が被らないようになるまで、4 番目の盤の各行ごとのシャッフルを繰り返す。

以下の図は、4 番目の盤まで生成が完了した図 3.4 である。

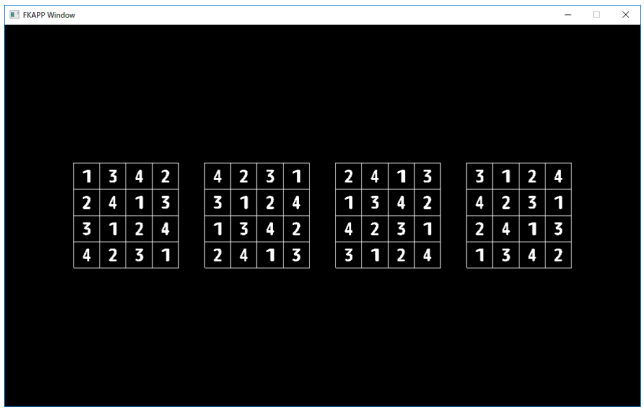

図 3.4　奥行き方向でも数独のルールが守られている

これで、4 × 4 × 4 の数独の解答が自動生成できたと言える。

3.5　4 × 4 × 4 の数独の問題の自動生成

生成した 4 × 4 × 4 の立体数独の解答から、ランダムなマスを選択し、そのマスの数字を 0 にする。このプログラムでは、0 を入力するとそのマスの表示が消えるようになっている。これを任意の回数だけ繰り返す。そうしてできた虫食い状態の盤面に、数独問題を解答するプログラムを実行する。この解答するプログラムは、0 が入力されているマスを検索し、そのマスに 1 から 4 を順に入れ、入力した数字が立体数独のルールを守っているかどうかをチェックするプログラムである。これを実行することで、その問題の解が 1 つになるかどうかのチェックを行うことができる。全てのマスの数字が確定し、解が一つになることが確認できれば、4 × 4 × 4 の立体数独問題の完成である。以下の図 3.5 は完成した 4 × 4 × 4 の立体数独の問題の問題の例である。

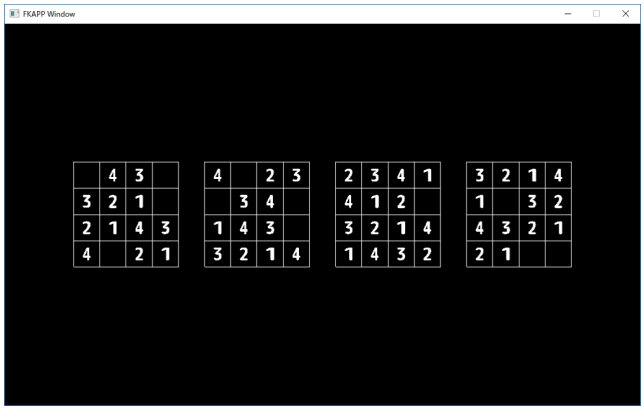

図 3.5　奥行き方向でも数独のルールが守られている

第 4 章

提案手法の検証と考察

提案した手法によって、問題が正確に自動生成できたかどうか確かめる。製作環境は Visual-Studio2017、ライブラリは FK[22] を使用した。検証方法としては、本研究で作成したプログラムを実行し、生成されたものを手動で解く。解くことができれば、生成は成功であると言える。同時に、実用的かどうかを調査するため、プログラムの実行時間を計測するコードを一時的に追加して、生成時間とそれぞれの盤面の生成回数も計測した。空きマス数の指定は 1 盤面につき 4 に設定し、検証は 10 回行った。検証環境を表 4.1 で示す。

表 4.1　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

以下の図 4.1 から図 4.10 が、実際のプログラムの実行結果である。

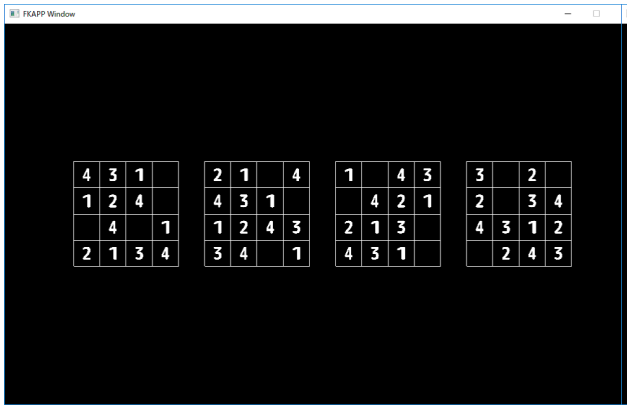

図 4.1　実験結果 1

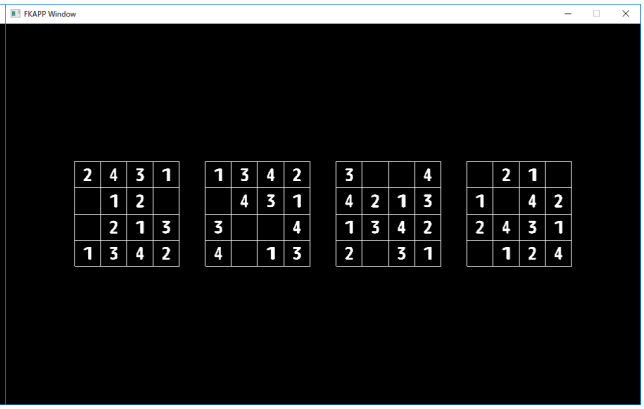

図 4.2　実験結果 2

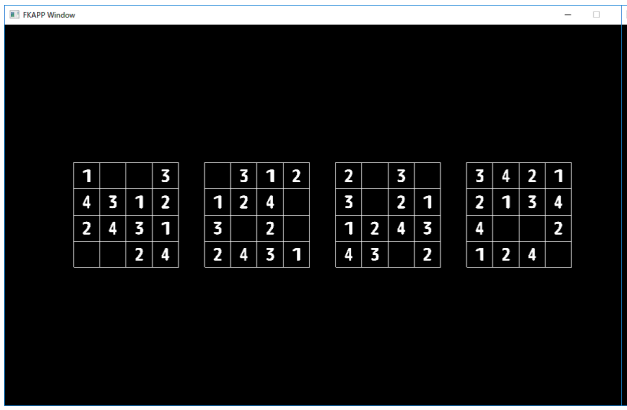

図 4.3　実験結果 3

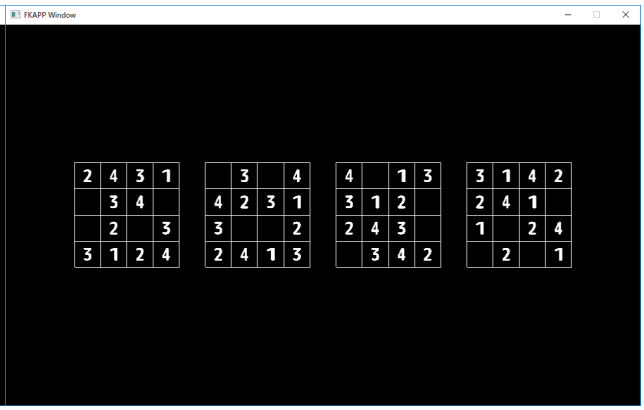

図 4.4　実験結果 4

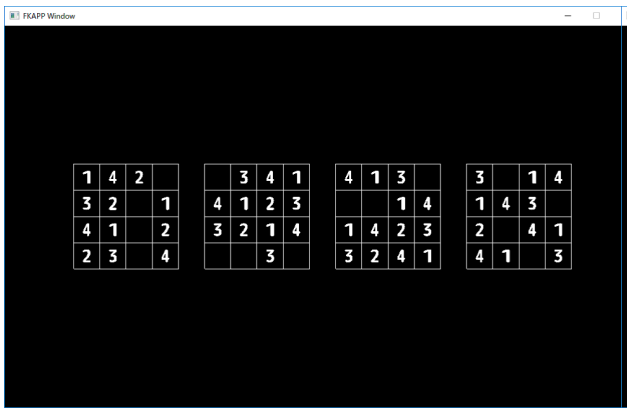

図 4.5　実験結果 5

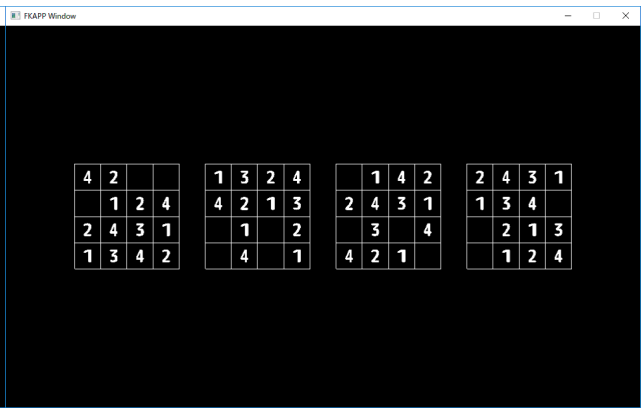

図 4.6　実験結果 6

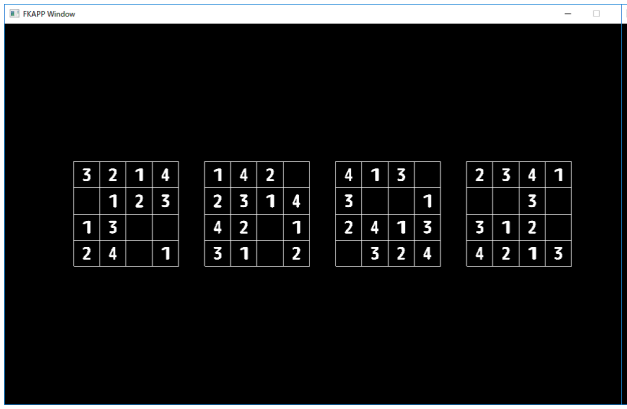

図 4.7　実験結果 7

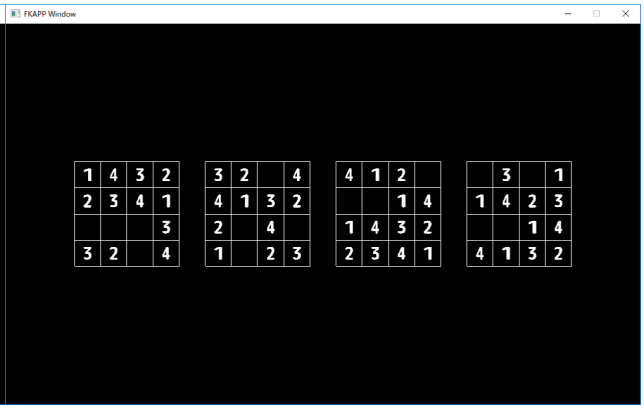

図 4.8　実験結果 8

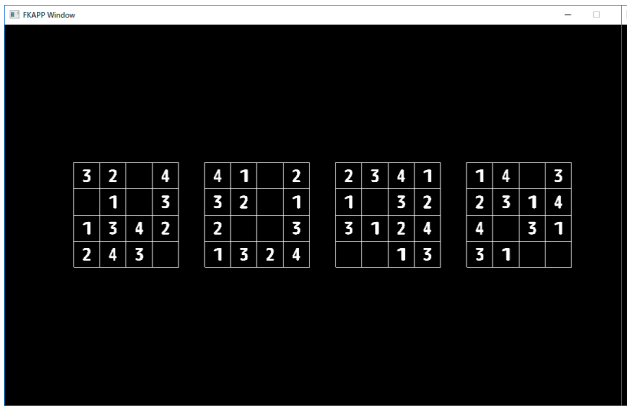

図 4.9　実験結果 9

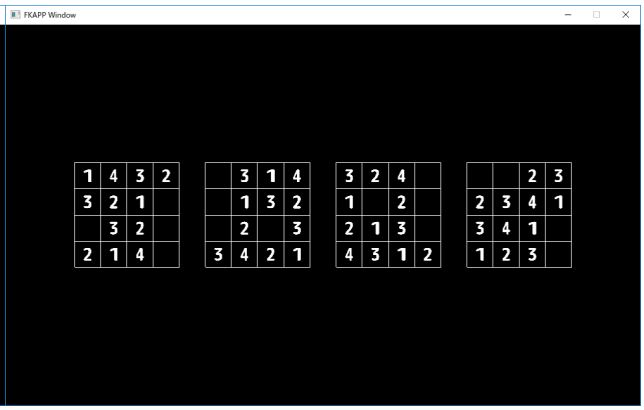

図 4.10　実験結果 10

次の表 4.2 は、それぞれの盤面の生成にかかった試行回数と、その合計数である。盤面 1 は一番左端の盤、盤面 2 は左から 2 番目の盤、盤面 3 は左から 3 番目の盤、盤面 4 は一番右端の盤である。シアン色で塗りつぶされたセルの数字が、盤面ごとの最小回数、黄色で塗りつぶされたセルの数字が、盤面ごとの最大回数である。また、表 4.3 は、それぞれの盤面の生成にかかった時間と、解答を生成するのにかかった時間、空白マスを生成し、問題を作成するのにかかった時間、それらの合計である。盤面 1 は一番左端の盤、盤面 2 は左から 2 番目の盤、盤面 3 は左から 3 番目の盤、盤面 4 は一番右端の盤である。シアン色で塗りつぶされたセルの数字が、盤面ごとの最短生成時間、黄色で塗りつぶされたセルの数字が、盤面ごとの最長生成時間である。すべての問題を人力で解答した結果、生成した問題が解けるものであり、かつ解が 1 つになることが確定して

いる問題であることが確認できた。また、検証の結果として、生成するまでにかかる試行回数や実行時間がかなり幅広くばらつくことが判明した。

表 4.2　生成回数に関する検証結果の詳細

	図名	盤面 1	盤面 2	盤面 3	盤面 4	合計
生成回数 (単位：回)	図 4.1	22	7,157	79,920	425,191	512,489
	図 4.2	315	51	57,498	760,091	817,955
	図 4.3	66	2,340	203,013	261,780	467,199
	図 4.4	616	10,144	140,895	55,403	207,058
	図 4.5	1,559	234	142,790	51,788	196,371
	図 4.6	1,255	13,452	13,333	149,978	178,018
	図 4.7	1,854	16,391	141,898	12,763	172,906
	図 4.8	222	14,075	151,080	25,308	190,685
	図 4.9	171	456	88,246	87,048	175,921
	図 4.10	3,237	110	301,853	464,634	769,834
平均生成回数		951.6	6.441	132052.6	229,398.6	512489

表 4.3　生成時間に関する検証結果の詳細

	図名	盤面 1	盤面 2	盤面 3	盤面 4	解答生成	問題生成	合計
生成時間 (単位：秒)	図 4.1	0.385	1.173	17.16	36.298	55.016	0.173	55.189
	図 4.2	0.94	4.788	2.808	60.167	68.703	0.166	68.869
	図 4.3	0.404	5.554	6.844	7.218	20.02	0.21	20.23
	図 4.4	0.154	1.509	8.646	0.472	10.781	0.154	10.935
	図 4.5	0.596	3.748	53.09	43.761	101.195	0.186	101.381
	図 4.6	0.34	26.444	2.02	1.104	29.908	0.171	30.079
	図 4.7	0.038	3.273	3.878	18.194	25.383	0.169	25.552
	図 4.8	1.177	11.674	40.761	67.098	120.71	0.188	120.898
	図 4.9	0.477	0.596	93.097	66.419	161.399	0.167	161.566
	図 4.10	0.004	4.017	30.538	63.604	98.163	0.172	98.335
平均生成時間		0.4541	6.2776	25.9652	36.4335	69.1278	0.1756	69.3034

4.1　考察

立体数独のルールを守りつつ、盤面を正しく生成することができた。また、1 問当たりの生成にかかる時間が、平均して約 69 秒と、手動で問題を生成するよりも早く生成できた。しかしながら、盤面を生成する回数がまばらかつ非常に膨大な回数になってしまった。これは、過去に廃棄した盤面と同一の盤面や盤面として成立しない盤面もとりあえず生成し、その後確認するという盤面生成のアルゴリズム的に問題があるためと思われる。このアルゴリズムのまま 9 × 9 × 9 の立体数独へ対応させるのは計算時間が膨大なものにならないか不安が残る結果となった。

第 5 章

まとめ

数独のバリエーションである立体数独への研究はほぼ行われていなかったため、本研究では、問題を自動生成することへ挑戦した。現在のあらかじめ行ごとに数字を順に入れて置き、それをFisherYates シャッフルアルゴルズムで行単位で入れ替えるという方法で盤面の生成する今の手法では、試行回数が膨大になってしまうため、時間がかかるという難点が発見できた。また、空きマスを作るアルゴリズムが、ランダムな場所を選択するという単純なものとなっている。これを改良することで、より空きマスの多い問題の生成や、問題としての完成度が高いものも生成できると考える。9 × 9 × 9 の立体数独への対応は、現状の手法のままでは、必要な計算時間が膨大になることから難しいと考えられるため、より効率的な盤面生成アルゴリズムの考案が必要になると考える。

謝辞

本研究を進めるにあたり、Fisher Yates Shuffle やプログラミングなどをご教授いただいた渡辺大地准教授、阿部雅樹助手、そして、数独に関しての様々なアイデアや、プログラミングについての知識を教えてくれた同研究室のメンバーに感謝します。

参考文献

[1] Gabriele Cirulli. 2048. https://gabrielecirulli.github.io/2048/. 参照:2018.1.13.

[2] Gakko Net Inc. Make10. https://itunes.apple.com/jp/app/4%E3%81%A4%E3%81%AE%E6%95%B0%E5%AD%97%E3%81%A710%E3%82%92%E4%BD%9C%E3%82%8C-%E6%95%B0%E5%AD%A6%E3%83%91%E3%82%BA%E3%83%AB-make10/id878908923?mt=8. 参照:2018.1.13.

[3] Wheblio. Weblio 辞書 数独. https://www.weblio.jp/wkpja/content/%E6%95%B0%E7%8B%AC_%E6%AD%B4%E5%8F%B2. 参照:2017.12.17.

[4] Frazer Jarvis Ed Russell. There are 5472730538 essentially different Sudoku grids... and the Sudoku symmetry group. http://www.afjarvis.staff.shef.ac.uk/sudoku/sudgroup.html. 参照:2017.12.17.

[5] 藤原博文. 「ナンプレ」パズルの良問を自動・大量生成する新システム. http://www.pro.or.jp/~fuji/numplace/twenty/. 参照:2017.1.14.

[6] 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. 情報処理学会第 70 回全国大会, pp. 4–799, 2008.

[7] 那須律政. 数独の少数ヒント問題の生成に関する研究. 学士学位論文, 高知工科大学 情報システム工学科, pp. 1–24, 2012.

[8] 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. 情報処理学会研究報告, Vol. 25, pp. 1–6, 2011.

[9] 棟方渚, 志水雅俊, 松原仁. 皮膚電気活動を用いた数独問題の難易度評価. 情報処理学会研究報告, Vol. 35, pp. 1–4, 2012.

[10] 井上真大, 奥乃博. 本質的に異なる数独解盤面の列挙と番号付け. 情報処理学会第 71 回全国大会, pp. 4–741, 2009.

[11] 谷尾祐香里, 越後宏紀, 上河恵理, 阿原一志. 入力画像に基づく数独問題自動生成システムの提案. *The 21st Game Programming Workshop 2016*, pp. 89–93, 2016.

[12] 立石匡, 湊真一. 二分決定グラフを用いた数独パズルの解探索と列挙. 情報処理学会第 70 回全国大会, pp. 5–253, 2008.

[13] 松尾康夫. 数独の推論規則と難易度に関する考察. 情報処理学会 研究報告, Vol. EC-5, pp. 1–6, 2006.

[14] 田中貴拓, 新谷幹夫, 岩穴口貴祥, 白石路雄 eb. 立体数独アプリケーションの開発. 芸術学会論文誌, Vol. 14, pp. 257–263, 2015.

[15] 奥村晴彦. C 言語による最新アルゴリズム辞典. 技術評論社, 日本, 1991.

[16] Fisher.Ronald.Aylmer.Sir and YatesFrank. *Statistical tables for biological, agricultural and medical research, edited by R.A. Fisher and F. Yates. 6th ed.* Edinburgh: Oliver and Boyd, イギリス, 1963.

[17] NIKOLI Co. Web ニコリ. `http://www.nikoli.co.jp/ja/iphone/sd_tutorial/`. 参照:2018.1.13.

[18] ミシチャン. ナンバープレース, 数独 解法まとめ. `http://www.geocities.jp/master_mishichan/basic.html`. 参照:2017.12.17.

[19] lessthanpanda. 知れば知るほど奥が深い。SUDOKU（ナンプレ）の解法まとめ. `https:`

//matome.naver.jp/odai/2134764369832089501. 参照:2017.12.19.

[20] 今井洋輔. 攻略!難問. 世界文化社, 日本, 2017.

[21] ネットワークシステム研究室: 坂本直志, 中嶋勇気. 『数独プログラミング』 「解答が 1 通り」の数独の問題を自動生成するプログラムの実装と性能評価 . http://sunluck.blog.fc2.com/blog-entry-127.html. 参照:2017.12.17.

[22] Fine Kernel Project. Fine Kernel ToolKit System. https://gamescience.jp/FK/index.html. 参照:2018.1.21.