モンテカルロ木探索による数独少数ヒント盤面の生成

那須 律政[1,a)]　松崎 公紀[2,b)]

ペンシルパズルのうちでも，数独に対してさまざまな研究が行われている．そのうち，最少ヒント問題とは，唯一解を持つ盤面を構成できる最少のヒント数はいくつかという問題である．通常のサイズが 9 × 9 の数独については，2012 年 1 月に McGuire らによって，ヒント数 16 の盤面がない，すなわち，最少ヒントが 17 であることが示された．GPCC の 2011 年問題では，サイズが 16 × 16 の数独に対する最少ヒント問題がとりあげられ，そのなかで白川によってヒント数 56 の盤面が発見された．著者らもこの数独の少数ヒント問題に対して研究を行っている．数独の少数ヒント盤面を効率良く生成するには，広大な探索空間の中から適切に探索を行う必要がある．著者らは，ゲーム分野で近年活発に研究されているモンテカルロ木探索手法をこの問題に適用し，少数ヒント問題を効率良く生成することを目指している．本論文では，数独少数ヒント盤面生成にモンテカルロ木探索手法を適用するにあたり必要であった工夫と，得られた少数ヒント盤面について報告する．

キーワード：数独, 最少ヒント問題, モンテカルロ木探索

Generation of Less-Clue Sudoku Boards by Monte-Carlo Tree Search

Norimasa Nasu[1,a)]　Kiminori Matsuzaki[2,b)]

Abstract: Among several pencil puzzles, Sudoku has been studied most intensively. A research problem on Sudoku is called the minimum number of clues problem: how many clues are necessary to construct a Sudoku board with a unique solution. For the usual 9 by 9 Sudoku, McGuire *et al.* showed on January 2012 that there is no 16-clue Sudoku board with a unique solution and the minimum number of clues is 17. In GPCC 2011, it was a problem to find a board with a smaller number of clues for the 16 by 16 Sudoku, and Shirakawa found a 56-clue board. We also have studied on generation of these less-clue Sudoku boards. A challenge in the effective generation of less-clue Sudoku boards is to find those boards from a very broad search space. We have applied the monte-carlo tree search technique, which has been widely studied in game programming, to generate efficiently less-clue Sudoku boards. In this paper, we show how we apply the monte-carlo tree search technique to the generation of less-clue Sudoku boards. We also report the Sudoku boards that we have obtained in the experiments.

Keywords: Sudoku, Minimum number of clues problem, Monte-carlo tree search

1. はじめに

数独は，ペンシルパズルの中でも世界的に人気のあるパズルであり，さらに，コンピュータ科学における研究対象としてもよく取り上げられるものである．例えば，汎用的なアルゴリズムやソルバを用いて数独を解く手法の研究 [6], [12], [13], [15] や，盤面の性質から難易度を推定する研究 [2], [8], [16], [17] が行われている．

本論文が関係する数独の最少ヒント問題は，唯一解を持つような問題を構成できる最少のヒント（表出数字）数がいくつであるかという問題である．9 × 9 の大きさの数独において，それまで 17 個のヒントからなる唯一解の盤面は知られていたが，McGuire らによりヒント数 16 の問題が存在しないことが示された [7]．16 × 16 のように，より大きな数独の場合の最少のヒント数がいくつであるかは未解決問題である．2011 年の GPCC では，この 16 × 16 の

[1] 高知工科大学大学院 工学研究科 基盤工学専攻
[2] 高知工科大学 情報学群
[a)] 165065d@gs.kochi-tech.ac.jp
[b)] matsuzaki.kiminori@kochi-tech.ac.jp

大きさの数独において，唯一解となるヒント数の少ない盤面を求めるという問題が出され，白川によりヒント数が 56 である盤面が得られている [3].

そのようなヒント数の少ない盤面を求めるのは容易ではない．その困難さの理由の 1 つは，ある条件を満たすような数独の盤面を生成するにあたり，その探索空間が非常に大きいことである．9 × 9 の盤面において 17 個の数をランダムに配置したとすると，（対称性を除去しないとして）その場合の数は 10^{25} にもなる．このような厖大な探索空間から，適切に目的の性質であるヒント数の少なくなるような盤面を探索する必要がある．もう 1 つの理由は，ヒント数の少ない盤面を表す特徴的なパラメータの設計が困難であることである．

本研究では，近年ゲームプログラミングの分野において活発に研究が行なわれているモンテカルロ木探索を利用する．モンテカルロ木探索は，乱数を用いたシミュレーション（これをプレイアウトと呼ぶ）によって近似解を求めるモンテカルロ法をゲーム木探索に組み合わせたものである．モンテカルロ木探索の特長は，探索空間が膨大であり盤面に対する評価関数を適切に作ることが困難であるようなゲームに対して有効であることである．この特長により，コンピュータ囲碁において勝率を劇的に向上させた [18]．上に述べたように，数独の少数ヒント問題を生成する場合にも探索空間と評価関数の問題があるため，モンテカルロ木探索を数独の少数ヒント問題に適用することは有望であると考えた．

本論文では，モンテカルロ木探索を利用して数独の少数ヒント問題を生成するアルゴリズムを提案・実装し，実験によりその効果を調査する．ランダムなプレイアウトを用いる単純なモンテカルロ木探索では，ヒント数の少ない問題をほとんど生成することができなかった．そのため，3 つの観点でアルゴリズムの改良を行った．本論文の貢献は以下の 3 点である．

- モンテカルロ木探索に基づく数独の少数ヒント問題生成アルゴリズムを示した．
- よりヒント数の少ない問題を生成するための改良点を 3 つ考案・実装した．
- 実験による評価を行い，最も良い結果として 300 万プレイアウトの間にヒント数が 18 である問題を平均 49 種類生成することができた．また，1 つの実験においてヒント数が 17 である問題の生成に成功した．

本論文の構成は以下の通りである．第 2 節では数独およびその最少ヒント問題について導入を行う．第 3 節では，モンテカルロ木探索のアルゴリズムを復習する．第 4 節では，数独の少数ヒント問題を生成するアルゴリズムとその実装における工夫を述べる．第 5 節では 2 段階での評価実験について説明し，その結果を報告する．第 6 節で関連研究を述べる．第 7 節でまとめと今後の課題を述べる．

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

図 1 数独における問題の一例

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

図 2 図 1 の問題の解盤面

2. 数独とその最少ヒント問題

数独とは，図 1 のようにマス目で区切られた正方形の盤面にいくつかの数字があらかじめ配置されたものから始め，マスに数字を書き込んでいき，図 2 のようにすべてのマスに数字を入れるパズルである．本論文では，最も一般的な 9 × 9 の数独のみを扱う．まず，数独の盤面に関係する用語を定義する．

ヒント 盤面にあらかじめ配置されている数字をヒントと呼ぶ．

ブロック 数独の盤面のうち，太線で区切られた 3 × 3 の領域をブロックと呼ぶ．

数独では，マスに入る数は次の条件を満たす必要がある．

条件 1 1 つのマスには 1 つの数字が入る．

条件 2a 各横列には 1〜9 の数字が 1 つずつ入る．

条件 2b 各縦列には 1〜9 の数字が 1 つずつ入る．

条件 2c 各ブロックには 1〜9 の数字が 1 つずつ入る．

この規則にしたがって，すべてのマスに数字を入れることができれば完成となる．

解盤面 上記の数独のルールを満たすようにすべてのマスに数字が入ったその盤面を解盤面と呼ぶ．

候補数字 空マスにおいて，上記の数独のルールもしくはその他の解法により取り得ないと判定された数を除いた，解の候補となる数字を候補数字と呼ぶ．

人が数独の問題を解く際には，上記の数独のルールとそれによって生まれる規則性を用いて，候補数字を絞り込む作業を行う．ルールから導かれる規則性は解法とも呼ばれる．本論文中にて用いる解法を以下に示す．

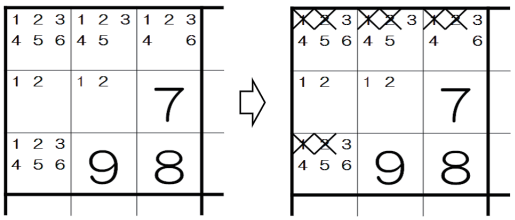

図 3 2-Naked による候補数字の絞り込み

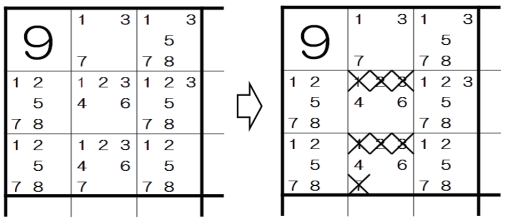

図 4 2-Hidden による候補数字の絞り込み

Cell-Unique あるマス x について，x が持つ候補数字が a 1 つだけであるとする．このとき，x の数字を a に確定する．これは，上記の条件 1 に対応する解法である．

Line-Unique ／ Block-Unique ある横列, 縦列, ブロックについて，数字 a が入り得るマスが x 1 つだけであるとする．このとき，x の数字を a に確定する．これは，上記の条件 2a，2b，2c に対応する解法である．

k-Naked ある横列, 縦列, ブロックについて, k 個のマス $x_1, x_2, \ldots, x_k$ が持つ候補数字が k 個の数 $a_1, a_2, \ldots, a_k$ の部分集合のみからなるとする．このとき，その横列，縦列，ブロックの $x_1, x_2, \ldots, x_k$ 以外のマスの候補数字から，$a_1, a_2, \ldots, a_k$ を消す．

k-Hidden ある横列，縦列，ブロックについて，k 個の数 $a_1, a_2, \ldots, a_k$ のいずれかを候補数字として持つマスが $x_1, x_2, \ldots, x_k$ の k マスのみであるとする．このとき，$x_1, x_2, \ldots, x_k$ の候補数字から，$a_1, a_2, \ldots, a_k$ 以外の候補数字を消す．

図 3 に解法 2-Naked を適用することで候補数字を絞り込む例を，図 4 に解法 2-Hidden を適用することで候補数字を絞り込む例を，それぞれ示す．

人が解くパズルであるためには，解盤面を求めるのに試行錯誤を必要としないという暗黙のルールがある．しかし，本論文では唯一解を持つことのみを数独の問題に対する条件とする．この条件の下で，数独の最少ヒント問題とは次の問題である．

最少ヒント問題 唯一解を持つような問題を構成できる最も少ないヒント数はいくつか．

本論文では，この最少ヒント問題に関連して，ヒント数の少ない問題を効率良く生成するためのアルゴリズムとその実装を示す．

```
MCT(root) {
  loop until 終了条件 {
    leaf <- select_downwards(root)
    leaf.n <- leaf.n + 1
    if (expand_cond(leaf)) {
      leaf <- expand(leaf).first_child
    }
    board = playout(leaf.board)
    update_upwards(leaf, getvalue(board))
  }
  return select_best_child(root)
}
```

図 5 モンテカルロ木探索アルゴリズム

3. モンテカルロ木探索

モンテカルロ木探索は，乱数を用いたシミュレーションにより近似解を求めるモンテカルロ法と，ゲーム木の探索とを組み合わせた手法である．モンテカルロ木探索では，ある盤面から終局まで乱数を用いてプレイすることを行う．これをプレイアウトと呼ぶ．

モンテカルロ木探索の重要なアイデアは次の 2 つである．1 つ目は，候補手に対してプレイアウトを行う際に，それまでに得られた評価値をもとに有望そうな候補手に多くのプレイアウトを割り当てることである．2 つ目は，ある候補手に対するプレイアウト回数がある閾値を超えた場合には，その手を展開してプレイアウトの開始点を 1 つ深くすることである．これらのアイデアにより，モンテカルロ木探索では有望そうな部分に対する探索を深く行うことができる．

図 5 にモンテカルロ木探索の擬似コードを示す．モンテカルロ木探索では，終了条件（探索時間もしくは探索回数）に到達するまでプレイアウトを繰り返し，最終的に評価値の最も良い候補手を選択する．繰り返しされる処理は，大きく次の 4 つからなる．

(1) それまでのプレイアウト回数と評価値をもとに，ルートから葉まで降りる（`select_downwards`）．
(2) 葉のプレイアウト回数を増やし，閾値を超えた（`expand_cond`）際にはそのノードを展開する（`expand`）．
(3) 選択された葉から，乱数によるプレイアウトを行う（`playout`）．
(4) プレイアウトの結果の盤面から評価値を計算し（`getvalue`），葉からルートまで値を更新する（`update_upwards`）．

図 5 において関数として記述した，プレイアウトの対象とする子ノードの選択方法 `select_downwards`，ノード展開の閾値 `expand_cond` と展開方法 `expand`，プレイアウト方法 `playout`，終局盤面の評価値 `getvalue` およびノードの持つ値を更新方法 `update_upwards` をそれぞれ定める

ことによりモンテカルロ木探索を行うことができる．得られた結果の善し悪し（ゲームプレイヤの場合はその強さ）は，これらのパラメータ関数をどのように定めるかに依存する．

以下では，コンピュータ囲碁などで広く用いられている UCT（Upper Confidence Tree）アルゴリズムで用いられている子ノードの選択方法について述べる．UCT では，子ノードの選択に UCB1（Upper Confidence Bound）アルゴリズム [1] を用いる．これは，多腕バンディット問題（Multi-armed bandit problem）[14] において，計算量が小さく高い報酬が得られる戦略である．UCB1 アルゴリズムでは，まず，各候補に対して次の式 1 で与えられる UCB1 値を計算する．式において，$\overline{X_j}$ は j 番目の候補の評価値の期待値，n_j は j 番目の候補のプレイアウト回数，n は全体のプレイアウト回数，c は問題ごとに定める定数である．

$$ucb_j = \overline{X_j} + c\sqrt{\frac{2\log n}{n_j}} \tag{1}$$

次に，得られた UCB1 値のうち最大となるような候補を選択する．この UCB1 アルゴリズムを用いることで，有望な候補に多くのプレイアウトを行うことと，評価値が偶然低い場合に対する救済という 2 つの性質によりプレイアウト対象が決定される．

4. 数独の少数ヒント問題生成プログラム

第 3 章で述べたとおり，モンテカルロ木探索をある問題に適用するためには図 5 に示したパラメータ関数を適切に定める必要がある．まず，本研究で作成した少数ヒント問題生成プログラムにおいて，それぞれの処理をどのように設計したかを述べる．その後，少数ヒント問題生成プログラムを作成する際に行った工夫を述べる．

4.1 少数ヒント問題生成へのモンテカルロ木探索の適用

少数ヒント問題生成プログラムにおいて，モンテカルロ木探索のパラメータ関数をどのように設計したかを述べる．

終局盤面の評価値

モンテカルロ木探索をゲームに適用する際に用いる評価値は，終局盤面もしくは終局盤面までに得られる得点とすることが多い．本研究では数独の問題盤面におけるヒント数を調べる対象としているので，最終的に唯一解となった問題盤面に含まれるヒント数を終局盤面の評価値とした[*1]．

子ノードの選択方法

本研究では，子ノードを選択する方法として，2 つの方法を設計した．

1 つ目の方法は，評価値が小さいものが良いとするよう，UCB1 値を単純に変更するものである．モンテカルロ木探索において，UCB1 は広く利用されている有益な子ノードの選択方法である．本研究では評価値をヒントの数としたため，評価値の小さなものがより良い選択となる．したがって，UCB1 値を与える式 (1) の第 2 項の符号を反転した式

$$v_j = \overline{X_j} - c\sqrt{\frac{2\log n}{n_j}} \tag{2}$$

（変数の意味は式 (1) と同じである）によって得られる値が最小となる子ノードを選択する．

解が唯一であるような盤面の中で，少数ヒント盤面は数が非常に少ない．したがって，評価値の期待値が評価値の最小値から大きく離れてしまうこともありうる．このことから，2 つ目の方法では，式 (2) の期待値 $\overline{X_j}$ の代わりに評価値の最小値 $\min(X_j)$ を用いる．

$$w_j = \min(X_j) - c\sqrt{\frac{2\log n}{n_j}} \tag{3}$$

```
playout(board) {
  loop until solve(board) == unique {
    board <- solveByRule(board)

    hint <- selectHint(board)
    board' <- addHint(board, hint)

    if (solve(board') == noAnswer) {
      board <- deleteCand(board, hint)
    } else {
      board <- board'
    }
  }
  return board
}
```

図 6 プレイアウトのアルゴリズム

プレイアウト方法

プレイアウトの概略を図 6 に示す．

プレイアウトにおける 1 ステップの計算は，配置するヒントの選択（`selectHint`）し，そのヒントを配置した盤面が解を持つかどうかの判定（`solve`）した後，解がない場合にはそのヒントを候補数字から消す（`deleteCand`），もしくは，解が存在する場合にはそのヒントを盤面に加える（`addHint`）のいずれかを行う．この 1 ステップの計算を，解盤面が唯一となるまで繰り返す．関数 `solve` がすべての盤面について，解を持たない（`noAnswer`），唯一解を持つ（`unique`），複数解を持つ（`multiple`）のいずれであるかを正確に判定できるとき，この `playout` 関数は解を持つ盤面を受け取り唯一解を持つ盤面を返す．関数 `solve` の実装については，第 4.4 節に示す．

配置するヒントの選択方法として 2 つ設計した．1 つ目の方法（完全ランダム）は，候補数字から等しい確率でラ

[*1] 後に述べるルールベースの解法を適用すると，プレイアウトにおいて配置するヒントの他にルールによって決定される数が生じるが，それらのルールによって決定された数は評価値のヒントとして数えない．

ンダムに選択するものである．2 つ目の方法（候補数字減少）は，候補数字からランダムに 3 つを選択し，それを配置したときに候補数字が最も少なくなるものを選択するものである．

また，各ヒントを配置する前に，ルールベースの解法を適用することで確実に決まるヒントもしくは候補数字を決定する処理を行うプログラムと行わないプログラムの 2 通りを用意した．ここで用いたルールベースの解法については，第 4.3 節で述べる．

ノード展開の閾値と展開方法

葉ノードの展開の基準として，プレイアウトがある一定回数行われた際に葉ノードを展開することとした．この回数は，木全体で同じ回数とした．

次に，葉ノードの展開方法について述べる．子ノードの数は最大 30 とした．プレイアウトにおいて配置するヒントの選択方法と同じ方法で 30 個の盤面を選択する．それらの盤面に対し第 4.2 節で述べる正規化を行い，同一であると判断された盤面を除去する．また，ここで作った盤面がそれまでに展開された盤面と同じ盤面となった場合には，すでにある盤面のノードを子ノードとした．したがって，ノードの構造は木ではなく DAG となる．

ノードの持つ値の更新方法

ノードの値は，プレイアウトごとに更新する．その際，プレイアウト対象となる葉ノードの選択するまでにたどったノードを逆順にたどり，その経路上のノードの値を更新する．

4.2 本質的に同じ盤面の除去

探索空間を小さくするのに，対称性に基づいて本質的に同じ盤面を除去することは非常に有効である．

数独の任意の盤面に対して，以下の操作を任意の順序および任意の回数適用して生成される盤面は，本質的に同じ盤面である [11]．

- 数字の入れ替え
- 盤面の転置
- ブロック行の入れ替え，ブロック列の入れ替え
- ブロック行内での行の入れ替え，ブロック列内での列の入れ替え

例えば，図 7 の盤面と図 8 の盤面は本質的に同じである．

ある 2 つの盤面が本質的に同じであるかどうかを完全に判定することは容易ではない．そこで，本研究では図 9 に示す正規化アルゴリズムを用いた．まず，ヒント数を基準として，行および列を並び換える（`sortRowColumn`）．ここでは，まず，ヒント数の多いブロック行が上に，ブロック行内でヒント数の多い行が上になるようにする．さらに，ヒント数の多いブロック列が左に，ブロック列内でヒントの多い列が左になるようにする．次に，並び換えした盤面と転置（`transpose`）した盤面のそれぞれに対し出現順に

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

図 7 ヒント数 17 の盤面

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

図 8 図 7 と本質的に同じ盤面

```
normalize(board) {
  board  <- sortRowColumn(board)
  boardT <- transpose(board)

  board  <- renumbering(board)
  boardT <- renumbering(boardT)

  if (hashCode(board) < hashCode(boardT)) {
    return board
  } else {
    return boardT
  }
}
```

図 9 盤面を正規化するアルゴリズム

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

図 10 図 7 と図 8 の盤面を正規化した盤面

番号付けを行う（`renumbering`）．このようにして作成された 2 つの盤面のうち，ハッシュ値をもとに一方を選択し，それを正規化された盤面とする．図 7 と図 8 の盤面に正規化アルゴリズムを適用すると，どちらも図 10 の盤面に正規化することができる．この正規化アルゴリズムは，本質的に同じ盤面を完全に判定することはできないが，比較的

```
solve(board) {
  cnf <- CNFOfBaseRule
  cnf <- append(cnf, toCNF(board))

  if (satisfiable(cnf)) {
    answer = findAnswer(cnf)
    cnf2 <- append(cnf, negate(answer))
    if (satisfiable(cnf2)) {
      return multiple
    } else {
      return unique
    }
  } else {
    return noAnswer
  }
}
```

図 11 SAT を用いた解探索アルゴリズム

高速に同一盤面を判定することができる．

4.3 ルールベースの解法による候補数字の削減

ルールベースの解法を適用して候補数字を絞り込めば，あるマスに入る数字が自明に決定することがある．数字が自明に決定するマスにヒントを配置することは明かに無駄である．したがって，これらのルールベースの解法を適用して，自明にマスに入る数字の決定もしくは候補数字を削減する処理を実装した．ここで適用する解法 [16] は，2 章で述べた Cell-Unique, Line-Unique, Block-Unique, k-Naked, k-Hidden とした．

4.4 SAT ソルバによる解探索

プレイアウトにおける解探索を行う `solve` 関数は，SAT ソルバ sat4j [5] を用いて実装した（図 11）．SAT ソルバは，連言標準形（CNF）で与えられた命題論理式（以下 CNF 式と呼ぶ）の充足可能性を判定するためのプログラムである．第 2 章で示した数独のルールと盤面に配置されたヒントを CNF 式として表し，それを SAT ソルバで解く．

`solve` 関数では，まず，数独のルールを表現する CNF 式に，盤面に配置されたヒントを表現する CNF 式を追加する．数独のルールを表現する CNF 式には，文献 [6] で extended encoding と呼ばれる手法を利用した．これは，各マス，横列，縦列，ブロックに数字が高々 1 つ，また，少なくとも 1 つあることを表現したものである．次に作成した CNF 式を SAT ソルバに渡す．CNF 式が充足可能でないときには解なし（`noAnswer`）である．CNF 式が充足可能であるときには，1 通りの変数割り当てを求め，それを否定したものを CNF 式に追加する．このようにして作った新しい CNF 式を SAT ソルバに渡し，それが充足可能であるときには複数解（`multiple`），充足不可能であるときには唯一解（`unique`）とする．

この `solve` 関数はプレイアウト中に複数回呼び出されるため，この関数の高速化は全体の高速化において非常に重要である．実装では，数独のルールを表現する CNF 式を事前に SAT ソルバに渡したものを準備し，その CNF 式の追加にかかる時間を削減した．また，高々 1 つという条件を指定するのに sat4j の `addAtMost` メソッドを利用した．これにより，数独のサイズ N（$N = 9$）に対し $N^3(N-1)$ 個の節を使用するのに比べより高速に解くことができた．

表 1 生成手法比較実験の結果

プレイアウト	評価値	ルール解法	ヒント $N = 18$	ヒント $N = 19$	ヒント $N = 20$
候補数字減少	最小	あり	41.5	2337.5	20551.8
		なし	1.3	331.8	6093.0
	平均	あり	0.0	147.3	16398.8
		なし	0.0	0.3	20.5
完全ランダム	最小	あり	0.0	26.5	1167.8
		なし	0.0	1.3	188.5
	平均	あり	0.0	0.3	13.3
		なし	0.0	0.0	0.0

5. 実験

5.1 問題生成手法の比較実験

第 4 章で，プレイアウトにおけるヒント選択の方法，子ノードの選択方法，およびルールベースの解法の有無について，それぞれ 2 種類の実装を示した．これらの有効性を調べるため比較実験を行った．

共通するパラメータとして，盤面サイズを 9×9，式 (2) と式 (3) における c の値を 1.0，プレイアウト回数を 1,000,000，最大子ノード数を 30，子ノードを展開するプレイアウト回数の閾値を 30 とした．プレイアウト手法の 2 種類，評価値の計算の 2 種類，ルールベースの解法の適用の有無のすべての組み合わせ 8 通りについて，それぞれ乱数の種を変更して 4 回実験を行った．ヒント数 N が 18，19，20 である盤面の生成数の平均を表 1 に示す．

プレイアウトにおいてヒントの配置を選択する方法では，候補数字が減少するものを選ぶ手法の方がヒント数の少ない盤面を多く生成できた．子ノードを選択する方法では，評価値の最小値を用いる式を使う方がヒント数の少ない盤面を多く生成できた．ルールベースによる解法を適用するかどうかについては，プレイアウト手法や子ノードの選択方法と比較すると効果は小さいが，ルールベースによる解法を適用する方がヒント数の少ない盤面が生成できた．プレイアウト手法について候補数字が減少するものを選択，子ノードの選択方法について評価値の最小値を用いる，ルールベースによる解法を行うという組み合わせでは，ヒント数が 18 である盤面を 41 盤面生成できた．さらに，表 1 には示していないが，この組み合わせでの盤面生成において，図 12 に示すヒント数が 17 である盤面生成に成功した．

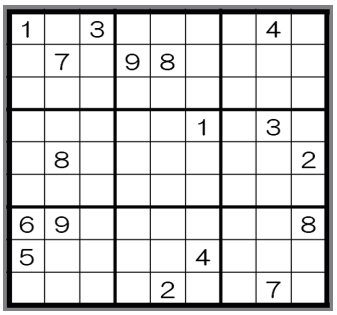

図 12 生成したヒント数 17 の盤面

表 2 モンテカルロパラメータ比較実験の結果．$N = 18$ の欄の括弧内は，ユニーク盤面の数．

c 値	閾値	$N = 18$	$N = 19$	$N = 20$
0.5	10	9.8 (8.5)	1058.5	18940.0
	20	103.0 (85.3)	4956.0	47937.8
	30	15.5 (13.5)	7749.8	44395.3
	40	23.0 (16.8)	3186.3	47136.8
1.0	10	80.5 (76.0)	5083.3	49454.0
	20	91.8 (81.3)	5496.5	60689.0
	30	23.5 (20.5)	3569.5	54060.5
	40	117.0 (90.5)	7469.8	78794.0

5.2 モンテカルロパラメータの比較実験

前節の実験によって，効率のよい生成手法を知ることができた．次に，モンテカルロ木探索のパラメータを変えて実験を行った．プレイアウト回数を 3,000,000，最大子ノード数 30 に固定し，式 (2) と式 (3) における c の値 0.5 と 1.0 の 2 通り，ノードを展開する閾値を 10，20，30，40 の 4 通りについてそれぞれ乱数の種を変更して 4 回問題生成実験を行った．ヒント数 N が 18，19，20 である盤面の生成数の平均を表 2 に示す．また，生成された $N = 18$ の盤面について，正規化を行い同じ盤面を除去したユニークな盤面数の平均を表 2 の括弧内に示す．

表 2 より，c 値が 1.0，閾値 40 の場合において平均で，ヒント数 18 の盤面を 117 盤面，うちユニークな盤面を 90 盤面生成することに成功し，これが最も多かった．c 値による違いを比較すると，全体として 0.5 よりも 1.0 の方が少数ヒント盤面の生成数が多いことから，c 値の違いはプレイアウトの精度に大きく影響を及ぼすことが確認された．一方，同じ c 値における実験を比較すると，閾値の変化と発見盤面数の違いに相関関係は見られない．これは，プレイアウト回数が十分な回数行われているため，木の成長速度による結果の差が無くなったためと考える．すなわち，プレイアウト回数を十分多い回数に設定した場合には，ノードを展開する閾値によってプレイアウトの結果に大きな差は生じないと言える．

6. 関連研究

6.1 数独に関する様々な研究

人が解くパズルとして数独は広く楽しまれている．計算機の場合，一般的な 9 × 9 の数独であれば，単純なバックトラックでも解くことができる．また，人が解く方法をシミュレートすることによりさらに高速に解くことができる．

このような従来のバックトラックとは異なるさまざまなアルゴリズムやソルバを使って数独を解けることも示されている．そのようなものとして，整数計画法を用いるもの [12]，二分決定グラフを用いるもの [15]，SAT ソルバを用いるもの [6]，GPU を用いた進化計算（遺伝アルゴリズム）によるもの [13] などがある．本研究では SAT ソルバを用いて数独の解を求めたが，第 4.4 節で述べたように盤面とルールの CNF 表現および SAT ソルバの使い方を工夫することで十分高速に計算することができた．

数独の問題を解くだけでなく，数独の問題や数独そのものの性質を調べる研究も行われている．例えば，人が解く際に用いる手法に着目することにより数独の問題盤面の難易度を評価する手法について，多くの研究がある [2], [8], [16], [17]．井上らは，盤面の対称性を除いた本質的に異なる数独の解盤面を数え上げている [11]．数独の最小ヒント問題について，McGuire らは 9 × 9 の数独において解が唯一となるヒント数 16 の盤面がないことを計算機で調べあげることにより示した [7]．より大きな数独においては最少のヒント数は分かっていないが，16 × 16 の数独においてヒント数が 56 である盤面と 25 × 25 の数独においてヒント数が 174 である盤面がそれぞれ白川によって発見されている [3]．

6.2 モンテカルロ木探索

モンテカルロ木探索 [4] は，探索空間が広大であり評価関数を設計することが難しいようなゲームにおいて有効なアルゴリズムである．その性質から，コンピュータ囲碁において強いプレイヤを作ることに大きく貢献している [18]．

これまでに，さまざまな種類のゲームに対してモンテカルロ木探索が適用されてきた．そのうち，1 人完全情報ゲームに対してモンテカルロ木探索を適用する研究として，SameGame へ適用することが Schadd ら [9] や Takes ら [10] によって行われている．本研究の少数ヒントの数独の問題生成は，唯一解となる盤面を解（終了状態），そのときのヒント数を得点とすれば 1 人完全情報ゲームとしてとらえることができる．すなわち，モンテカルロ木探索を 1 人完全情報ゲームへ適用する 1 つの応用として捉えることもできる．

7. まとめ

本論文では，数独においてヒント数の少ない問題を効率よく生成することを目標として，ゲーム研究で活発に研究されているモンテカルロ木探索を数独の問題生成に適用した．完全にランダムなプレイアウトを行う単純なモンテカルロ木探索では効率よくヒント数の少ない問題を生成することができなかったため，モンテカルロ木探索を改良する必要があった．本論文では，プレイアウト時に候補数字が少なくなるようにヒントを配置すること，複数の子を持つノードの評価値を計算する際に平均ではなく最小値を利用すること，プレイアウト中に人の解き方を使って決まるところを確定すること，の 3 つ改良方法を提案し，実際に実装して評価実験を行った．3,000,000 回のプレイアウトからなる実験を行った結果，ヒント数 18 の問題を平均で 49 種類見つけることができた．また，実験の中でヒント数が最少の 17 である盤面も 1 つ発見することができた．

今後の課題としては大きく次の 2 点が挙げられる．1 つ目は，プレイアウトにおけるヒントの置き方や評価値の計算法を改良することである．第 5.1 節の実験の結果に見られる通り，プレイアウトにおけるヒントの置き方は，問題生成の効率に大きな影響がある．また，本論文ではプレイアウト結果の評価値をヒント数のみを用いた．これらを改良することにより，少数ヒントの問題をより効率良く生成することができるのではないかと考えており，ヒント数 17 の問題が安定して生成できるようにすることが目標である．2 つ目は，2011 年の GPCC の問題であった 16 × 16 の数独へ本プログラムを適用し，すでに発見されているヒント数 56 の盤面よりもヒント数の少ない盤面を生成することである．すでに行った予備実験では，1 回のプレイアウトに約 1 秒程度かかってしまったため，プログラムの高速化が課題である．

参考文献

[1] P. Auer, N. Cesa-Bianchi, and P. Fischer: Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, Vol. 47, No. 2–3, pp. 235–256, 2002.

[2] M. Danburg-Wyld: How to Score Sudoku. http://sudokuplace.com/sudoku_scoring.asp.

[3] Games and Puzzles Competitions on Computers (GPCC): 数独のヒント最少問題の解答. http://hp.vector.co.jp/authors/VA003988/gpcc/11p1.htm, 2011.

[4] L. Kocsis and C. Szepesvári: Bandit Based Monte-Carlo Planning. In *Proceedings of the 17th European Conference on Machine Learning (ECML 2006)*, pp. 282–293, 2006.

[5] D. Le Berre and A. Parrain: The Sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, Vol. 7, No. 2–3, pp. 59–64, 2010.

[6] I. Lynce and J. Ouaknine: Sudoku as a SAT problem. In *Proceedings of 9th International Symposium on Artificial Intelligence and Mathematics*, 2006.

[7] G. McGuire, B. Tugemann, and G. Civario: There is no 16-Clue Sudoku: Solving the Sudoku Minimum Number of Clues Problem. *The Computing Research Repository (CoRR)*, abs/1201.0749, 2012.

[8] R. Pelánek: Difficulty Rating of Sudoku Puzzles by a Computational Model. In *Proceedings of the Twenty-Fourth International Florida Artificial Intelligence Research Society Conference*, 2011. AAAI Press.

[9] M. P. D. Schadd, M. H. M. Winands, M. J. W. Tak, and J. W. H. M. Uiterwijk: Single-player Monte-Carlo tree search for SameGame. *Journal Knowledge-Based Systems*, Vol. 34, pp. 3–11, Elsevier, 2011.

[10] F. W. Takes and W. A. Kosters: Solving SameGame and its Chessboard Variant. In *Proceedings of the 21st Benelux Conference on Artificial Intelligence (BNAIC)*, pp. 249–256, 2009.

[11] 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, **一般社団法人情報処理学会 全国大会講演論文集**, 平成 21 年, No. 4, pp. "4-741"–"4-742", 2009.

[12] 岡本 吉央: 「整数計画法によるパズル解法」実習報告. **組合せゲーム・パズルミニプロジェクト第 *2* 回ミニ研究集会**, 2007.

[13] 佐藤 裕二, 長谷川 直広, 佐藤 未来子, 並木 美太郎: メニーコアプロセッサによる進化計算を用いた数独解法の高速化. **情報処理学会研究報告**, *HPC,* **ハイパフォーマンスコンピューティング**, Vol. 130, No. 4, pp. 1–7, 2011.

[14] 但馬 康宏, 小谷 善行: k-armed バンデット問題のゲームへの適用における試行回数について. **情報処理学会研究報告**, *AL,* **アルゴリズム研究会報告**, Vol. 2008, No. 49, pp. 65–70, 2008.

[15] 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. **一般社団法人情報処理学会 全国大会講演論文集**, 平成 20 年, No. 5, pp. "5-253"–"5-254", 2008.

[16] 土出 智也, 真貝 寿明: 数独パズルの難易度判定 －解法ロジックを用いた数値化の提案－. **大阪工業大学紀要, 理工篇**, Vol. 56, No. 1, pp. 1–18, 2011.

[17] 松原 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**, *EC,* **エンタテインメントコンピューティング**, Vol. 2006, No, 134, pp. 1–6, 2006.

[18] 美添 一樹: モンテカルロ木探索 : コンピュータ囲碁に革命を起こした新手法. **情報処理**, Vol. 49, No. 6, pp. 686–693, 2008.