Monte Carlo Methods Appl. **16** (2010), 251–263
DOI 10.1515/MCMA.2010.018

Generalizing Sudoku to three dimensions

Tiffany A. Lambert and Paula A. Whitlock

Abstract. The well-known logic puzzle Sudoku can be generalized from two to three dimensions by designing a puzzle that is played on the faces of a cube. One variation, already introduced as a puzzle by Dion Church, uses three adjacent faces. Another variation uses all six faces. We have developed a set of rules and constraints for both three-dimensional Sudoku variations and have studied the properties using the method of simulated annealing.

Keywords. Sudoku, Simulated annealing, stochastic games, Markov chains.

2010 Mathematics Subject Classification. 65C40, 91A15.

1 Introduction

Sudoku is a logic puzzle consisting of a $n^2 \times n^2$ grid of cells partitioned into n^2 $n \times n$ blocks. When completed, every row, column and block in the puzzle must contain the numbers 1–n^2 just once. A Sudoku puzzle comes with some cells assigned values, called fixed cells, and the empty cells are to be filled in by the player. Published puzzles usually have $n = 3, 4$ or 5 and the rules can be summarized as:

- Each row of cells contains the integers 1 through n^2 exactly once.
- Each column of cells contains the integers 1 through n^2 exactly once.
- Each $n \times n$ block of cells contains the integers 1 through n^2 exactly once.

The general problem of solving Sudoku puzzles is known to be NP-Complete [1], and many methods have been developed to obtain optimal solutions. The Sudoku examples most people are familiar with are called *logic-solvable* because a logical chain of reasoning will usually lead to a solution. But there also exist puzzles whose solution can only be found by guessing a random solution or by applying brute force iteration through all possible combinations. Calculating exactly how many unique puzzle solutions exist is an interesting combinatoric problem [2, 3].

As the popularity of Sudoku has grown, so has the number of playing variations. There have been several attempts to make Sudoku more challenging [1, 4]. A particular version, published several years ago, was Dion Church's attempt at three-dimensional Sudoku, shown in Figure 1, which is played on three faces of

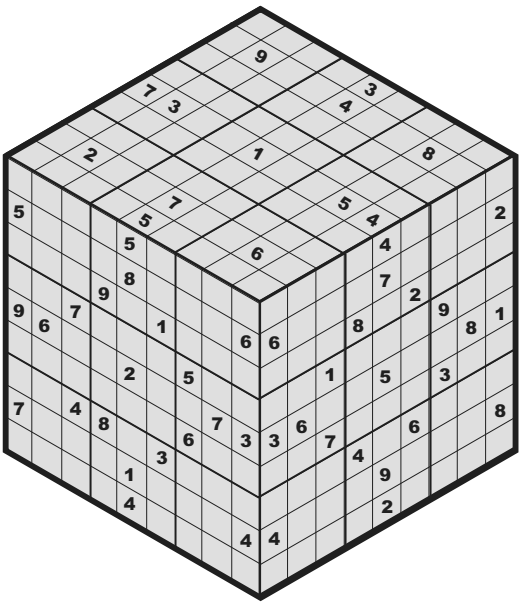

Figure 1. The original three face Dion Church puzzle.

a cube [5]. In the original published version of the puzzle, each face followed the traditional rules for the Sudoku puzzle. In addition, the cells on the edges of each face matched.

This paper describes the analyses of puzzles similar to the one proposed by Dion Church. We also discuss the extension of the puzzle to the six faces of a cube. We label both these versions three-dimensional Sudoku.

2 3D Sudoku puzzles

2.1 The three face model

Our three-dimensional Sudoku models have a much larger state space than a regular two-dimensional Sudoku puzzle. In our first variation with three faces, each face is an individual puzzle and follows the traditional rules of Sudoku. The size of the individual blocks on each face studied here may have either $n = 3$ or $n = 4$. The latter will yield a face that is a 16×16 grid. The edges where the faces meet require additional constraints. We considered both the original Dion Church puzzle constraints which stipulate that the cells on the face edges must match as shown in Table 1 and the alternative, shown in Table 2, that the cells on the edges must differ.

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

Table 1. A sample edge of two faces where the cell values match when $n = 4$.

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

Table 2. A sample edge of two faces where the cell values do not match when $n = 4$.

2.2 The six face model

Our extended version of a three-dimensional Sudoku puzzle considers the six faces of a cube. Opened out to two dimensions, the puzzle can be displayed as in Figures 2 and 3. For simplicity, completed puzzles with a block size of 2×2 are shown. The puzzle in Figure 2 has the constraint that cells on the face edges do not match in value. Figure 3 illustrates the alternative constraint that puzzles on the edges do match. While the diagrams in Figures 2 and 3 seem to imply that the faces could be considered independently, this is, of course, untrue. Every face is dependent on four other faces.

The techniques for solving Sudoku puzzles are many and use such approaches as logical search methods [6], constraint programming [1], and genetic algorithms [7]. Because the three-dimensional state space is large, a general and efficient method was needed. Lewis [8,9] used simulated annealing [10] to investigate the properties of two-dimensional puzzles. This algorithm searches for the optimal solution to a problem by using Monte Carlo methods [11].

3 Simulated annealing applied to solving Sudoku puzzles

With simulated annealing, an optimization problem is organized to use a cost function, $U(X)$, to measure how close a proposed solution is to an optimal solution. The probability distribution of the proposed solutions is given by

$$F(X) \propto e^{-\lambda * U(X)} \tag{3.1}$$

where λ is labeled an "inverse temperature." The cost function has either a maximum or a minimum when an optimal solution is located. The search algorithm begins at a "high temperature" which allows large excursions into the initial search space. As the cost function value changes, the value of λ changes according to a cooling schedule.

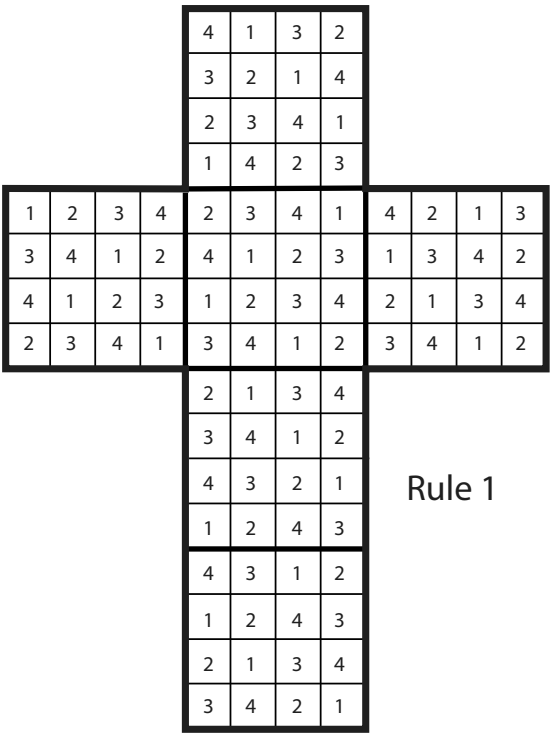

Figure 2. A complete six face puzzle when $n = 2$. In this case the edge cells do not match.

In Lewis' approach [8] for the two-dimensional puzzle, all empty cells of the initial puzzle are filled in randomly within each block. The random values are chosen from 1 to n^2 where the fixed cell values in the block have been removed from the list. The cost function is then computed as the sum of the values in the cells of each row and column. In a completed, correct puzzle the contribution to the sum per row or column is $(n^2 + 1) * n^2/2$ and this is subtracted from the cost function. A solution has been found when $U(X) = 0$. The algorithm proceeds by proposing a random swap between two non-fixed cells in a block and the cost function is recalculated. The swapped values are always accepted if the cost has decreased. However, this iterative improvement may get stuck in a local minimum rather than finding the optimal solution. Simulated annealing, therefore, uses the Metropolis algorithm [12] as a means to escape local minima. The latter algorithm will accept a move probabilistically that increases the cost function. The number

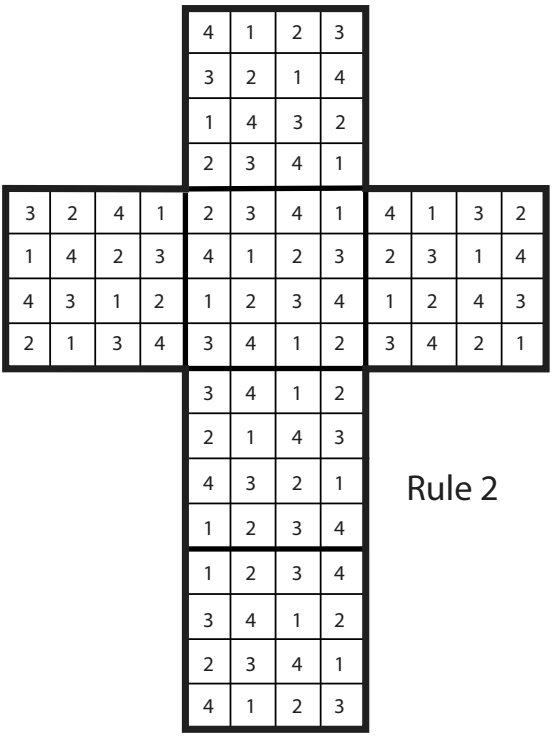

Figure 3. A complete six face puzzle when $n = 2$. In this case the edge cells match.

of swaps attempted at each temperature, a Markov chain, is related to the number of empty cells. At high temperature, most of the swaps of cells are accepted. At a low temperature, only the most favorable swaps are accepted. The cost function approaches a minimum, 0, as the algorithm approaches a solution.

Lewis's simulated annealing method is capable of producing a solution for both partially filled, as well as completely empty grids. Thus, it is not only a solver, but also a generator for all size Sudoku grids.

3.1 Application to the Dion Church Sudoku puzzle

We adapted the simulated annealing method and Lewis's code to solve the Dion Church three faces of a cube puzzle. The two-dimensional puzzle was represented in the simulated annealing code as a two-dimensional array of values. To ac-

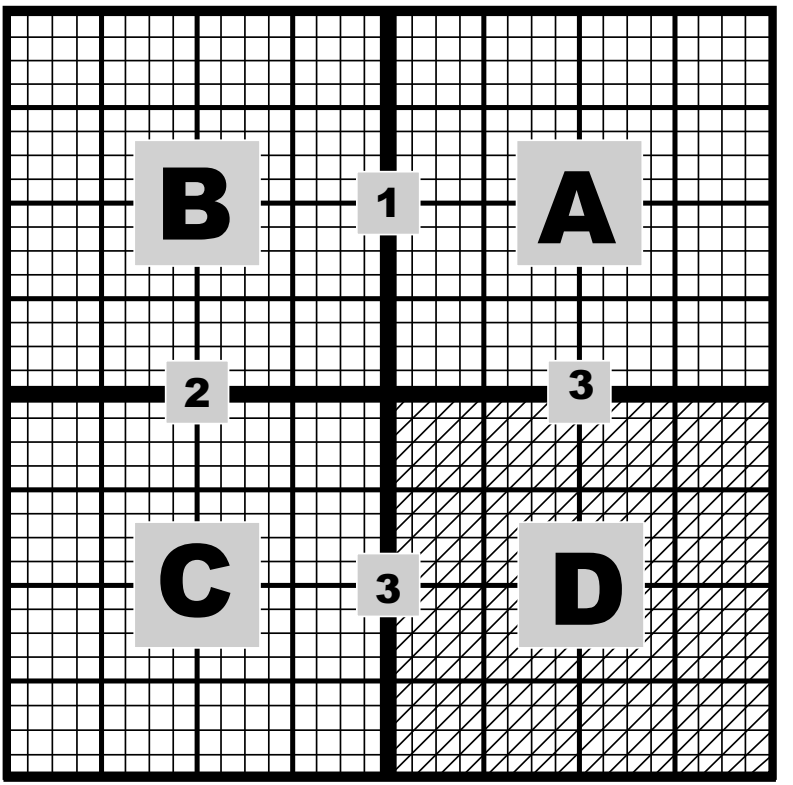

Figure 4. Implementation of the 3-face puzzle in the simulation.

commodate the three-dimensional puzzle, the array was expanded to a size of $2n^2 \times 2n^2$. The array was divided into four quadrants, one for each face, with the fourth quadrant unused, see Figure 4. The edge constraints were implemented so that the edges correspond as follows:

- Edge A1 adjoins edge B1
- Edge B2 adjoins edge C2
- Edge C3 adjoins edge A3

In order to configure the edge constraint into the cost function, $U(X)$, in Eq. (3.1), we added an edge contribution to the row and column cost. For every edge cell that fails the constraint condition, we added one point to the total edge cost function. This amount is consistent with the column and row sums and has a sufficient influence in accepting or rejecting a move.

The six faces of a cube puzzle was implemented as two parallel $2n^2 \times 2n^2$ arrays with the added constraints that every edge is dependent on another face. The Markov chain of moves involves swaps in both the parallel arrays. Furthermore, every time an edge cell is swapped, its value must be compared with an edge cell of the neighboring face. This made the determination of a solution very slow.

3.2 The cooling schedule

In the two-dimensional case [8], the temperature was decreased by 1 % after the annealing process equilibrated, i.e. completed one Markov chain of moves, at each inverse temperature. We refer to this as a 99 % cooling rate. It worked well for the two-dimensional 3×3 grids, but in the larger three-dimensional puzzle with edge constraints, the constant cooling schedule was no longer efficient. Increasing the change in temperature to 5 % (95 % cooling rate), 10 %, or 15 % did improve the efficiency of the code, but increased the occurrence of freezing into non-optimal solutions. This led to experiments with dynamic cooling schedules in which the temperature decreases by a larger amount initially, and then at a much slower rate as the system approaches a solution. To achieve this, we made the cooling rate a function of the cost, so as the cost function decreases from its initial state, so does the change in temperature. The resulting algorithm is greedy and much more efficient.

In Table 3, the average times using constant percentage changes in the temperature are shown. Reheats to a high temperature were performed if the algorithm could not find a solution for a run within the allocated time. Reheats were allowed to occur three times before the algorithm terminated the search. Both sets of edge constraints had similar behaviors.

Increasing the size of the temperation change from 1 % to 20 % did lead to faster calculations. However, from Table 3, it is clear that a cooling rate less than 70 % causes the system to cool too quickly and not find a solution. The run times for several dynamic cooling schedules are shown in Table 4, where the cooling rate is dependent on the change in the overall cost function. The corresponding

	50 %	60 %	65 %	70 %	75 %	80 %	85 %	90 %	95 %	99 %
Avg (sec)	562	383	296	103	55	43	92	80	166	853
Std. Dev.	n/a	n/a	n/a	207	71	24	100	100	26	135
Reheats	yes	yes	yes	yes	yes	no	yes	no	no	no

Table 3. Timings for solving a 3-face Sudoku puzzle with constant cooling rates. n/a indicates that a solution was not found.

	A	B	C	D
Avg (sec)	286	58	116	29
Std. Dev.	171	59	151	13
Reheats	yes	yes	yes	no

Table 4. Timings for solving a 3-face Sudoku puzzle with dynamic cooling schedules.

A		B		C		D	
change in cost	cooling rate	change in cost	cooling rate	change in cost	cooling rate	change in cost	cooling rate
0	50	0	85	0	50	0	50
25	65	25	80	25	55	25	60
50	80	50	70	50	65	50	70
75	90	75	65	75	70	75	75
90	95	90	55	90	80	90	80
95	99	95	50	95	85	95	90

Table 5. The changes in cooling rate as a function of the change in the cost function from the initial value for different cooling schedules.

schedules are shown in Table 5. The timings with the dynamic cooling schedules are dependent on the initial cooling rate, the change in the cost function from the initial value, and the final cooling rate. Trial and error led us to cooling schedule D which has been the most efficient. This schedule improved the run time by more than 90 % from the initial constant cooling schedule.

The three face algorithm can easily generate many unique Sudoku puzzles starting with an empty array and given a different pseudorandom number sequence. If the code is initialized with a partially filled in grid, it will find the unique solution or if there is more than one possible solution, it will find at least one of them in most cases. A completed puzzle generated by the simulated annealing code is shown in Figure 5. The 0's represent the nonparticipating cells in our representation of the puzzle.

4 Results

In the discussion that follows, the focus will be on the three face puzzle. The six face puzzle was solvable but took a very long time even for $n = 3$.

Logic solvability is still a question that is being explored. There are many papers employing different methods of solving Sudoku, but little work has been done

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

Figure 5. The internal representation of a completed three face puzzle in the simulated annealing code. The lower right quadrant is not part of the puzzle.

on the generation of puzzles. It has not been proven, but the smallest known number of fixed cells in a 3×3 that yield a unique Sudoku solution is 17 [4]. Moreover, it appears that at least 30 % of the cells need to be fixed for the puzzle to be logic-solvable [8]. This idea is much harder to conceptualize in a three-dimensional space. Computing the minimum number of givens for a unique solution is an optimization problem that is NP-complete.

However, we can empirically study how many fixed cells are needed to produce a unique puzzle. Using a completed three face puzzle, m cells were systematically emptied, i.e. $m = 5, 10, 15, \ldots$, their values removed, and the puzzle was resolved by the code with a different sequence of pseudorandom numbers. If the new solution was identical to the original puzzle after several repetitions of the process, the solution was deemed unique. For $n = 3$, if less than 100–120 cells are empty, that is 123–143 cells are fixed, the puzzle has a unique solution. However, preferentially removing the values in cells on common edges leads to multiple possible solutions when more than 45 cells are empty. For $n = 4$, solutions are unique when less than 260 cells are empty. Another issue involves which cells need to be fixed to maintain logic solvability by using a chain of reasoning. Sudoku solving programs exist that use strictly logical and/or human solving heuristics to both solve puzzles, and can be used to determine if the puzzle is logic solvable [6].

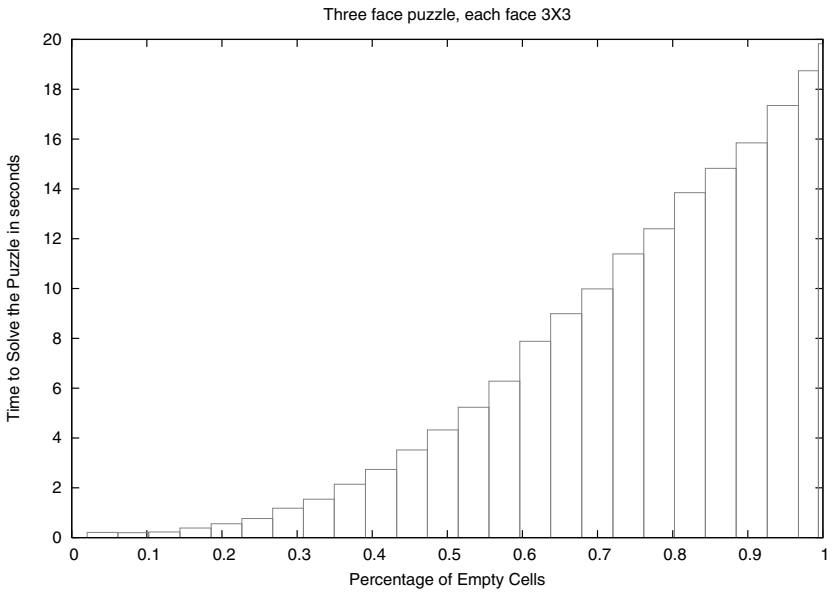

Figure 6. Average time to solve a three faces of a cube Sudoku puzzle when $n = 3$.

A fascinating aspect of using simulated annealing to find solutions to Sudoku puzzles is the existence of what has been called a "phase transistion" in the time to solve a puzzle as the number of initially fixed cells changes. This behavior was first observed by Lewis [8] in the two-dimensional version of the puzzle as the value of n was increased from 3 to 5. We have performed similar experiments with the three faces of a cube puzzle. As the percentage of empty cells changes from almost 0 (an almost complete puzzle) to 100 % (a puzzle with no fixed cells), the time to solve the puzzle increases. This reflects the fact that the search space for the almost complete puzzle is very small and increases as the percentage of empty cells increases. For $n = 3$, the increase in time is monotonic, see Figure 6. The time to solve a three face puzzle is shown versus the percentage of empty cells. Each bar represents the average time for trying to solve 20 independent puzzles. However, when $n = 4$, there are cases where no solution may be found even though the algorithm is allowed to execute for a long time. This is shown in Figure 7, where the arrows indicate cases where one or more of the puzzles were not solved in over a half hour of computing time.

That such cases exist is not surprising since the two-dimensional puzzles have been shown to be NP-complete. This very interesting phenomenon becomes harder to study as n increases as the time to solve the puzzle increases dramatically. For $n = 3$, a three faces of a cube puzzle starting with all cells empty, takes only

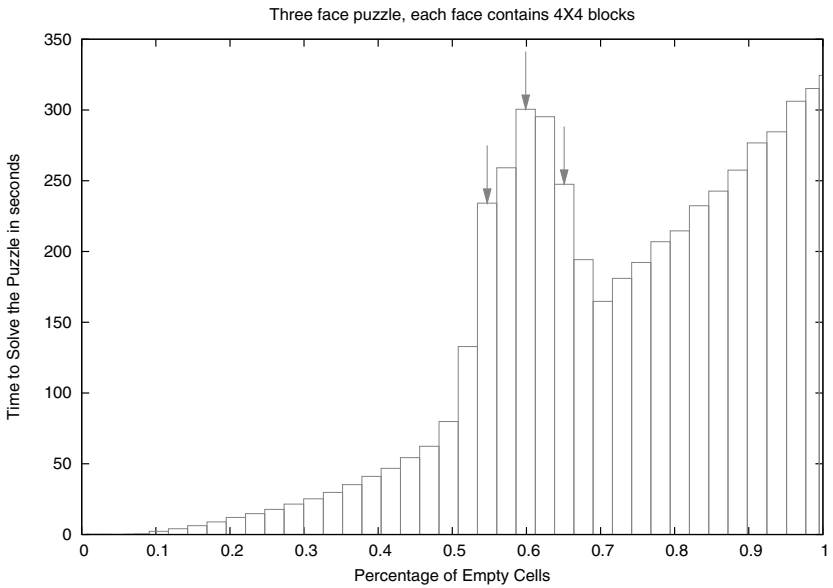

Figure 7. Average time to solve a three faces of a cube Sudoku puzzle when $n = 4$.

19 seconds to find a solution. When n is increased to 4, an initially empty puzzle takes over 325 seconds to find a solution. The case of $n = 5$ takes 20 times as long to solve compared to a $n = 4$ puzzle. We were unable to complete a study of the phenomenon with the six faces of a cube puzzle due to the enormously longer solution times.

5 Conclusions

From our experiments with the cooling schedule, we have determined that both changes in the cost function and the cooling rate are key factors in finding an optimal solution. As the difference between the initial cost and the current cost becomes larger, the cooling rate should become slower in order to restrict jumps to a much less optimal solution. By changing the cooling rate dynamically, we reduced the number of Markov chains that are started, thus decreasing the run time.

Trial and error was used to determine how many fixed cells need to be present to yield a unique solution. For $n = 3$, when cells were removed randomly from a completed puzzle, the percent of fixed cells was found to be at least 59 %. For $n = 4$, the percent of fixed cells needed increased to 67 %.

Experiments with the three face puzzle showed that the "phase transition" in the search time to find a solution observed in the two-dimensional puzzle is still present. As the search space becomes larger and the number of possible solutions increases, the time to find a solution dramatically increases. This occurs when the number of fixed cells decreases from 50 to 30 %. In some experiments with $n = 4$, no solution was found after several reheats of the algorithm.

To investigate the properties of the six face puzzle the simulated annealing algorithm needs to be made more efficient, perhaps by parallelizing the code. For example, multiple swaps could be performed in parallel (ensuring that the swapped cells are non-overlapping) and increase the efficiency of finding a solution.

An interesting extension of the current research is the relationship between the solving of Sudoku puzzles and the construction of point sets. It has been pointed out by Mullen [13] that sets of mutually orthogonal Latin squares can be used to construct (t,m,s)-nets and in some cases, the nets may have optimal properties. It has been suggested that the search by simulated annealing to find an optimal solution to a Sudoku puzzle could be related to finding (t,m,s)-nets with optimal parameters.

Acknowledgments. One of us, T. A. L., acknowledges the Louis Stoke Alliance for Minority Participation and the National Science Foundation for providing us with a grant. We were also partially supported by PSC/CUNY Award 61519-00 39. We thank Prof. Murray Gross for recommending that we study generalized Sudoku puzzles. We thank Dr. Rhyd Lewis for sharing his code for solving two-dimensional Sudoku puzzles by simulated annealing and Dr. Alyssa Lees for her insights. We also thank Dr. Gottlieb Pirsic for pointing out the possible relationship between solving Sudoku puzzles and (t,m,s)-nets during the 7th IMACS Seminar on Monte Carlo Methods (Brussels, September 2009).

Bibliography

[1] G. Santos-Garcia and M. Palomino, Solving Sudoku Puzzles with Rewriting Rules, *Electronic Notes in Theoretical Computer Science*, **176** (2007), 79–93.

[2] B. Felgenhauer and F. Jarvis, Mathematics of Sudoku I, *Mathematical Spectrum*, **39** (2006), 15–22.

[3] E. Russell and F. Jarvis, Mathematics of Sudoku II, *Mathematical Spectrum*, **39** (2006), 54–58.

[4] L. Taalman, Taking Sudoku Seriously, *Math Horizons*, **15** (2007), 5–9.

[5] D. Church, 3D Sudoku, *The Daily Telegraph* (May 2005).

[6] S. K. Jones, P. A. Roach and S. Perkins, Construction of Heuristics for a Search-Based Approach to Solving Sudoku, in *Proc. of AI-2007, the* 27^{th} *SGAI Int'l. Conf. on Innovative Tech. and Appl. of AI*, ed. M Bramer, F. Coenen and M. Petridis, Springer, London, (2008) 37–39.

[7] T. Mantere and J. Koljonen, Solving and rating Sudoku puzzles with genetic algorithms, *Proceedings of the 12th Finnish Artificial Intelligence Conference*, (2006), 86–92.

[8] R. Lewis, Metaheuristics can solve Sudoku puzzles, *Journal of Heuristics* **13** (2006), 387–401.

[9] R. Lewis, On the Combination of Constraint Programming and Stochastic Search, in *Hybrid Metaheuristics, Lecture Notes in Computer Science*, **4771**, Springer, Berlin, (2007), 96–107.

[10] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, Optimization by Simulated Annealing, *Science*, **220** (1983), 671–680.

[11] M. H. Kalos and P. A. Whitlock, *Monte Carlo Methods, 2nd edition*, Wiley-VCH Verlag GmbH & Co., Weinheim (2008).

[12] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, Equation of state calculation by fast computing machines, *Journal of Chemical Physics*, **21** (1953) 1087–1092.

[13] G. L. Mullen, Combinatorial Methods in the Construction of Point Sets with Uniformity Properties, *Mathematics and Computer Modelling*, **23** (1996), 1–8.

Received November 13, 2009; revised September 8, 2010.

Author information

Tiffany A. Lambert, Department of Computer and Information Sciences,
Brooklyn College, 2900 Bedford Avenue, Brooklyn, New York 10021, USA.

Paula A. Whitlock, Department of Computer and Information Sciences,
Brooklyn College, 2900 Bedford Avenue, Brooklyn, New York 10021, USA.
E-mail: `whitlock@brooklyn.cuny.edu`