A New Algorithm for Generating Unique-Solution Sudoku

Article · January 2008

DOI: 10.1109/ICNC.2008.788

5 authors, including:

Yue Wu
University of Oxford
18 PUBLICATIONS **41** CITATIONS
SEE PROFILE

Fourth International Conference on Natural Computation

A New Algorithm for Generating Unique-solution Sudoku

Baochen SUN Xiwei SUN Yue WU Yilong YIN Gongping YANG

Shandong University
baochens@gmail.com

Abstract

This paper describes a new algorithm for generating unique-solution Sudoku puzzles. Distinguished from common algorithms, it guarantees a unique solution itself rather than relaying on some unique-solution test algorithms. What is more, the time complexity of our algorithm is polynomial, which is a significant progress as most of the generation algorithms are non-polynomial.

1. Introduction

Sudoku, an abbreviation for a Japanese phrase, meaning "the digits must remain single", is usually a puzzle presented on a square grid with 9*9 entries, consisting of nine 3*3 regions such that every row, column and region should be filled with the digits 1 to 9. An example follows:

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

Figure1. Sudoku puzzle

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

Figure2. Answer for the puzzle above

Through researching existed relevant information, all the generating algorithms are based on solving algorithms of the puzzle. We present a much more direct generating algorithm, which don't rely on the solving algorithms to guarantee the uniqueness of solution.

2. Previous work

Generally, there are two approaches available, both based on the solving algorithms, for the puzzle generation:

• Starting with an empty Sudoku square and add preset values one by one until meet some metrics.

• Staring with a full-filled Sudoku square and remove values until meet some metrics.

When use those algorithms we need to test whether the new puzzle has a unique solution as every entry added or moved. This process of common algorithms relies on some unique-solution test algorithms, which are mostly based on the solving algorithms of the Sudoku puzzle.

3. A new generating algorithm

3.1. Terminologies and symbols

Strategies Strategies are hints in a Sudoku puzzle, which can be concluded by applying reasoning. The simplest strategy is to determine the candidates of blank entries according to the constraints of row, column and regions. There are lots of strategies of different levels, such as "Naked pair", "Hidden triple", "Swordfish" [1], etc.

N_C The number of candidates that can be filled in one entry and do not cause any inconsistent temporarily.

N{C} Suppose C is one set, then function N{C} shows the number of elements in C.

Chain A loop, formed by straight lines that link entries with some two values from 1 to 9, make a chain. An example follows:

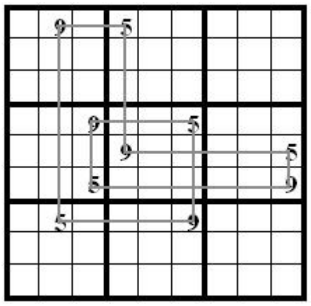

Figure3. A chain consist of 5s and 9s

We also have some relevant conclusions:

• Every two values from 1 to 9 in a certain full-filled Sudoku square have at least one chain.

• All the straight lines that link entries filled with some two values, form chains exactly.

• Suppose in a full-filled Sudoku square, we moved all the entries of a given chain. As long as one entry refilled, sometimes even with no entry refilled, the chain could be recurred by reasoning.

3.2. Puzzle generate algorithm

We designate the 81 entries of a given Sudoku square as the universal set S, and another pair of subsets which are complementary: one is named S_1 while the other is S_2.

In addition, S_1 is divided into two complementary subsets: S_{11} represents all the filled entries in S_1 while S_{12} represents all the blank entries in S_1. If we put some filled entries into S_{12}, it indicates these entries are blank automatically. Similarly, if we put some empty entries into S_{11}, it indicates these entries will be filled again with the initial values. There is one important property of S_1: **the entries in S_{11} can uniquely determine the value of all the entries in S_{12}.**

Essentially, our algorithm starts with a full-filled Sudoku square. Unlike the common algorithms, we do not need unique-solution test algorithms. We generate a Sudoku puzzle through expanding S_1 to the universal set S. As the property of S_1, when S_1 equals S a unique-solution Sudoku puzzle generated.

3.2.1. Pseudocode of the generation algorithm

```
1.  First search for the longest chain in the given
    square, and put all 18 entries filled with the
    related two values into $S_1$. Suppose that these
    entries also form j chains besides the initial chain.
    We withhold the value of one entry and empty all
    the other entries in each of (j+1) chains. Then
    there are (j+1) entries in $S_{11}$, and (18-(j+1))
    entries in $S_{12}$.
2.  Then randomly, we put another 9 entries of some
    value into $S_{12}$. To guarantee the property of $S_1$,
    we move some entries from $S_{12}$ to $S_{11}$ by
    applying greedy eliminating algorithm (explained
    below).
3.  Do step 1 to 2 till $S_1$ =S.
```

3.2.2 Pseudocode of the greedy eliminating algorithm

We set up a temporary set S' to store all the entries whose values can be determined by strategies.

```
1.  Determine candidates of each entry in $S_{12}$ by
    applying strategies.
2.  If $N_C$ of some entry is 1, move it to S' and refill
    the entry, turn to step 1.
3.  Find the entry whose $N_C$ is the most, move it to
    $S_{11}$, turn to step 1.
4.  When N {$S_{12}$} =0, move all the entries in S' to
    $S_{12}$ and the algorithm ends.
```

Based on Greedy technique, we can conclude that N {S_{12}} increasing during running the greedy eliminating algorithm.

As the generation algorithm executes N {S_1} =18+9*i after running i times of greedy eliminating algorithm, so the process eventually leads to N {S_1} =81. A unique-solution Sudoku puzzle therefore generated.

3.3. Proof of the correctness of the generation algorithm

Initialization: Initially, N {S_1} =18, and so the invariant is trivially true.
Maintenance: When each iteration the generation algorithm runs, N {S_1} increased by 9. Because of the greedy eliminating algorithm, each time N {S_1} increased the property of S_1 is warranted.
Termination: At termination, N {S_1} =81. Thus, a unique-solution Sudoku puzzle generated.

3.4. Complexity of the generation algorithm

Though the data of the algorithm is very small, it is safe that we do not consider the space complexity.

The time complexity of the inner algorithm—the greedy eliminating algorithm is polynomial since the time complexity of applying each strategy is polynomial. And the times of executing the outer algorithm is constant (8 times), so the overall time complexity of this algorithm is polynomial.

3.5. Metrics of difficulty

We can use many metrics to define the difficulty levels, such as the number of blank entries and the position of the blank entries [1]. As the generation algorithm terminates, when the most difficult puzzle generated, we should refill some entries according to the difficulty level needed.

3.6. Extension

When each iteration the algorithm described above runs, N {S_1} increased by 9. We can easily expand the algorithm that each iteration N {S_1} increased by 9*k (k is a natural number that lower than 7). By comparisons of those algorithms, we can find the most feasible algorithm.

4. Conclusion

In this paper we present a brand-new algorithm for generating unique-solution Sudoku puzzles. Its complexity is practicable and it is very easy to expand.

5. References

[1] Wei-Meng Lee, *Programming Sudoku*, Apress, Berkeley, CA, USA, 2006
[2] Thomas H. Cormen, Charles E. Leiseison, Ronald L. Rivest, Clifford Stein, *Introduction to Algorithms, Second Edition*, MIT press, Cambridge, MA, USA, 2001