Solving Sudoku Puzzles Based on Customized Information Entropy

Gaoshou Zhai[1] and Junhong Zhang[1,2]

[1]*School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China*
[2]*School of Electrical and Information Engineering, Beijing University of Civil Engineering and Architecture, Beijing 100044, China*

gszhai@bjtu.edu.cn, zhangjunhong@bucea.edu.cn

Abstract

Conception and calculation method of information entropy is customized for Sudoku puzzles and a corresponding algorithm is designed to solve Sudoku puzzles. The definitions of inverse information entropy and information amount for inverse information entropy are also introduced and directly used instead of information entropy in order to simplify the solving procedure. Experimental results show that the algorithm has better time efficiency than available methods including generic algorithms and rule based algorithms and it can solve not only unique-solution puzzles (including extremely difficult puzzles) but also multiple-solution puzzles. It is concluded that information entropy can be used for grading Sudoku puzzles as well.

Keywords: *Sudoku puzzle, Solving algorithm, Customized information entropy, Inverse information entropy, Information amount for inverse information entropy*

1. Introduction

Sudoku puzzles can trace back to ancient Chinese JiuGongGe (*i.e.* 9-Palace Grid) puzzles, but their truly similar original form is thought as Latin squares, which is invented by Swiss mathematician Leonard Euler in 18 century. Furthermore, the first modern Sudoku puzzle was created by an American retired architect named Howard Garns and published in the May 1979 edition of *Dell Pencil Puzzles and Word Games*. And the game owes its popularized success to Wayne Gould, a peripatetic retired judge living in Hong Kong, who came across it while visiting Japan in 1997 and wrote a computer program that automatically generates Sudoku grids and persuaded the London *Times* of publishing the puzzles at the end of 2004 followed by several daily papers in countries all over the world [1].

Nowadays, the game becomes one of the most popular games that are helpful to improve intellectual development in the world. And many scholars are attracted to develop solving algorithms based on computers [2-7].

A standard Sudoku puzzle is a flat square grid containing 81 cells well-distributed in 9 rows and 9 columns and the grid is divided into 9 smaller squares called by subgrids which contain 9 cells each. In general, the game begins with numbers already printed in some cells. And the player must fill in all the empty cells with the numbers 1 to 9 in such a way that no digit appears twice in the same row, column or subgrid. Some typical Sudoku puzzles are

illustrated in Figure 1, where (A) is a Sudoku in French final games [2], (B) is a difficult Sudoku that **iSudoku** can't solve [2] and (C) is a Sudoku with 34 solutions [3].

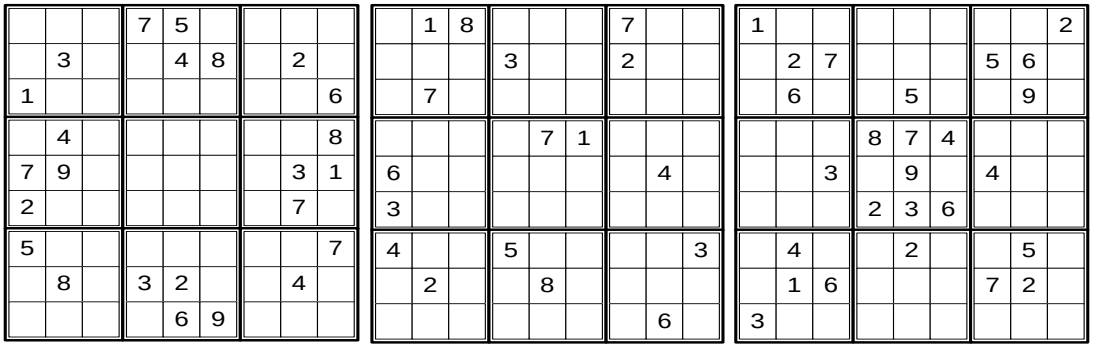

Figure 1. Some Typical Sudoku Puzzles

There are mainly three types of methods for computer programs to solve Sudokus. Initially, searching method based on backtracking is used to solve Sudokus [4]. But for its clumsiness, some experts turn to imitate human logic inference capability or to exploit some manual experiences in programs for solving such problems. Thereafter, some rule based methods are put forward [2, 3]. In addition, genetic algorithm, evolutionary algorithm, cultural algorithm, simulated annealing algorithm and other artificial / computational intelligence methods are also used to solve Sudokus [5-7].

All in all, searching method based on backtracking have the advantage of all-purpose feature and it is widely used in many web Sudoku solvers. On the other side, methods based on rules or based on computational intelligence have the disadvantage of time-consuming or become incapable of solving some difficult Sudoku puzzles. For these reasons, searching method based on backtracking is focused in this paper and information entropy is introduced to reduce useless attempt of candidate numbers and corresponding inference paths so as to improve the searching efficiency.

2. Preliminary

2.1. Entropy and Information Entropy

The term entropy was coined in 1865 by a German physicist named Rudolf Clausius. And he developed the thermodynamic definition of entropy by describing how to measure the entropy of an isolated system in thermodynamic equilibrium. While an Austrian physicist named Ludwig Boltzmann developed the statistical definition of entropy in the 1870s by analyzing the statistical behavior of the microscopic component of the system and he demonstrated equivalence of the above two definitions of entropy. In simple words, entropy is a measure of disorder in the universe. Now it is applied in cybernetics, life science and many other disciplines.

Claude Elwood Shannon, the father of information theory, published his paper "A Mathematical Theory of Communication" in 1948 and introduced the concept of entropy in

information theory that is the measure of the amount of information missing before reception. In another word, it is a measure of the uncertainty associated with a random variable. Furthermore, if a variable is more uncertain, entropy is greater while the amount of information required for fixing it is greater too.

2.2. Customized Information Entropy for Sudoku

For the cells in Sudoku grid that are empty without beforehand given or afterhand fixed number, they are in fact just like variables. The player can fill them according to the numbers in those cells that are in the same row, column or subgrid with each empty cell and these numbers are just like information required for fixing those variables corresponding to empty cells. Accordingly, if no cell that are in the same row, column or subgrid with a specified empty cell is given or fixed unique number, the specified empty cell must be most uncertain and its entropy can be assigned a maximum number, *e.g.* 1. On the contrary, if a cell is given a number in advance or fixed a number in the inference procedure, the cell is undoubtedly determinate and its entropy can be assigned a minimum number, *e.g.* 0. Because there are at most 9 results or 9 candidate numbers for each cell, its information entropy can be described as follows.

Definition 1 The information entropy of a Sudoku cell is a measure of the uncertainty of the cell and it can be assigned **x**/9 where **x** represents the amount of positively supportive information i.e. possible candidate numbers for the cell in the current state.

Note that they are different cases that a cell is filled with a number (no matter beforehand or afterhand) and an empty cell with only one possible candidate number by inference. As to the former situation, the state of the cell is confirmed to be certain, and the set of candidate numbers for it can be thought as empty. But for the latter situation, the state of the cell is still uncertain, in another word, the state of the cell isn't certain until it has been filled with the unique candidate number. In addition, numbers 1-9 are all candidate numbers for any empty cell at the beginning of the solving procedure.

Definition 2 The information entropy of a Sudoku puzzle is a measure of the uncertainty of the Sudoku grid and it can be assigned **SEG**/81 where **SEG** represents the sum of entropy for all cells in the Sudoku grid.

The concept of the information entropy of a Sudoku cell is used to decide which empty cell should be firstly processed and fixed in the solving procedure while the concept of the information entropy of a Sudoku puzzle can be used to measure the difficulty level of a Sudoku puzzle as will be further discussed in the following sections about difficulty grading. Moreover, the empty cell having the least information entropy (so it is in the state farthest to the uncertainty) ought to be firstly processed and fixed in the solving procedure.

2.3. Inverse Information Entropy and Information Amount for Inverse Information Entropy

Because candidate numbers for an empty cell are figured out according to those numbers filling in the cells that are in the same row, column or subgrid with the empty cell, the

definitions of inverse information entropy and information amount for inverse information entropy can be introduced as follows in order to make the solving procedure more convenient and simpler.

Definition 3 The inverse information entropy of a Sudoku cell is a measure of the certainty of the cell and it can be assigned **y**/9 where **y** represents the amount of oppositely supportive information (i.e. excluded numbers for the cell in the current state), which can be called information amount for inverse information entropy (abbr. IAIIE).

Obviously, the sum of **x** and **y** is 9. Accordingly, the information entropy of a Sudoku cell plus the inverse information entropy of the same Sudoku cell in the same state equals 1.

Typically, for the case that a cell is filled with a number (no matter beforehand or afterhand), it means that the excluding inference procedure is over and the state of the cell is confirmed to be certain. In another word, all the numbers 1-9 have been attempted to exclude and the IAIIE for the cell is 9. Meanwhile, for an empty cell at the beginning of the solving procedure, no number has been excluded and the IAIIE for the cell is 0.

In the solving procedure, the empty cell with the maximum IAIIE (so it has the least information entropy) will be firstly selected to be processed.

3. Methodology

The method to solve Sudoku puzzles based on information entropy includes three key aspects, i.e. initialization and synchronization of IAIIEs, straightforward solving procedure and backtracking based solving procedure.

3.1. Initialization and Synchronization of IAIIEs

Because the IAIIEs of cells are used to decide which empty cell ought to be firstly processed, correct calculation of IAIIEs is crucial for solving Sudoku puzzles.

Above of all, the IAIIEs of all cells are assigned 0 from beginning and they are initialized in turn according to the Sudoku puzzle itself. Moreover, if the number of a cell is given in advance, the IAIIE of the cell ought to be assigned 9. At the same time, if it is the first time for the fixed number of a cell to be excluded from candidate numbers of any empty cell that is in the same row, column or subgrid, the IAIIE of the empty cell ought to be synchronized and added by 1. This is so-called synchronization of IAIIEs.

During the solving procedure, once an empty cell is filled with a number regardless of whether it is solely decided or it is just one of possible candidate numbers, its IAIIE ought to be assigned 9 while the IAIIEs of all empty cells that are in the same row, column or subgrid with it ought to be synchronized as above also. Most importantly, if the IAIIE of some cell is 9 after synchronization but the cell itself is still empty, the former attempt of filling number in cell has to be thought as failure and backtracking ought to be executed.

3.2. Straightforward Solving Procedure

It is well-known that a blind rigid backtracking based algorithm is time-consuming. So a straightforward solving procedure is introduced to solve those simpler Sudoku puzzles that

can be solved without backtracking in order to improve program efficiency. At the same time, the procedure will be called by backtracking based solving procedure to solve a relatively difficult Sudoku puzzle that can't be solved only by the straightforward solving procedure itself. Furthermore, as for a Sudoku puzzle, the straightforward solving procedure is firstly used to attempt to solve it and the backtracking based solving procedure will be used once the attempt fails.

The basic idea of the straightforward solving procedure is to figure out empty cells whose IAIIEs are 8 in turn. It can get the final or staged solution of a Sudoku puzzle.

Given that *bSolved* is a Boolean variable that represents whether the Sudoku puzzle has been solved completely and *bGoAheaded* is another Boolean variable that represents whether the solution has been advanced in the current loop of while (*bSolved*=false). In addition, integer loop variables *i* and *j* are used to index the row and column for each cell of Sudoku grid respectively while *zR* is used to set the return value of the straightforward solving procedure.

The straightforward solving procedure can be described as follows:

step 0. Set *bSolved* = false;

step 1. If *bSolved* = false, go to step 2; else set *zR* = 1 and go to step 12;

step 2. Set *bSolved* = true, *bGoAheaded* = false and *i* = 0;

step 3. If *i* < 9, go to step 5; else go to step 4;

step 4. If *bGoAheaded* = true, go to step1; else go to step 11;

step 5. Set *j* = 0;

step 6. If *j* < 9, go to step 7; else set *i* = *i*+1 and go to step 3;

step 7. If the value of *Cell[i][j]* is 0, go to step 8; else set *j* = *j*+1 and go to step 6;

step 8. If the *IAIIE* of *Cell[i][j]* is 8, go to step 9; else set *j* = *j*+1 and *bSolved* = false and go to step 6;

step 9. Set the value of *Cell[i][j]* by its unique candidate number and let the *IAIIE of Cell[i][j]* = 9, synchronize the IAIIEs of all cells that are in the same row, column or subgrid with *Cell[i][j]* and check whether succeed;

setp 10. If synchronization of IAIIEs failed, set *zR* = -1 and go to step 12; else set *j* = *j*+1 and *bGoAheaded* = true and go to step 6;

setp 11. If *bSolved* = true, set *zR* = 1; else *zR* = 0;

setp 12. Terminate and return *zR*.

Programming logic for the straightforward solving procedure can also be illustrated in Figure 2.

Note that there are three possible cases of return value for the straightforward solving procedure. If it returns -1, backtracking ought to be executed in its caller, *i.e.* the backtracking based solving procedure. If it returns 1, a final solution is got. If it returns 0, a staged solution

is got and the caller, *i.e.* the main routine or the main function, ought to call the backtracking based solving procedure so as to advance the solving process further and to get the final solution(s).

3.3. Backtracking Based Solving Procedure

Backtracking based solving procedure is used to solve those Sudoku puzzles that can't be solved only by the straightforward solving procedure. In addition, it can get all the solutions for those Sudoku puzzles that have multiple solutions.

Given that *zSolutionNo* is a static integer member variable to be used to count solutions for the Sudoku puzzle and *zLoop* is another integer variable to be used as a loop variable so as to traverse all the candidate numbers for Cell[i][j].

The backtracking based solving procedure can be described as follows:

step 0. Set *zSolutionNo* = 0;

step 1. Copy the current state of Sudoku grid including values, IAIIEs and candidate number sets for all cells to a temporary object variable with the same class, e.g. *zTemp*;

step 2. Search the cell that has the maximum IAIIE and suppose that the result is *Cell[i][j]*;

step 3. Construct the candidate number set for *Cell[i][j]* and suppose its size is *zSize*;

step 4. Set *zLoop* = 0;

step 5. If *zLoop* < *zSize*, go to step 6; else go to step 13;

step 6. Set the value of *Cell[i][j]* by its *zLoop*-th candidate number and let the *IAIIE of Cell[i][j]* = 9;

step 7. Synchronize the IAIIEs of all cells that are in the same row, column or subgrid with *Cell[i][j]* and check whether succeed;

step 8. If synchronization of IAIIEs failed, go to step 12; else go to step 9;

step 9. Call the **Straightforward Solving Procedure** and store its return value at *zResult*;

setp 10. If *zResult* = 1, set *zSolutionNo* = *zSolutionNo* + 1 and output the *zSolutionNo*-th Sudoku solution and go to step 12; else go to step 11;

setp 11. If *zResult* = 0, call the **Backtracking Based Solving Procedure** itself;

setp 12. Restore the state of Sudoku grid by *zTemp* and set *zLoop* = *zLoop* + 1;

setp 13. Terminate.

Programming logic for the backtracking based solving procedure can also be illustrated in Figure 3.

3.4. Acquiring the Execution Time of Algorithm

Execution time of algorithm ought to be got in order to compare time performance among different algorithms. And many functions are provided in C/C++ language that can be used to

get execution time (refer to Table 1, wherein 1-4 methods are based on Windows series platforms and 5-7 methods are based on Linux /Unix platforms).

Table 1. Methods of Computing Execution Time Provided in C/C++ Languages

No	Prototype and description for corresponding functions
1	*clock_t clock(void);* timing is accurate to 1/CLOCKS_PER_SEC second (e.g. 1ms in VC++6.0)
2	*time_t time(time_t * timer); double difftime(time_t time2, time_t time1);* timing is accurate to 1s
3	*DWORD GetTickCount(void);* timing is accurate to 1ms and limits to 49.7 days
4	*BOOL QueryPerformanceFrequency(LARGE_INTEGER ***lpFrequency**);* *BOOL QueryPerformanceCounter(LARGE_INTEGER *lpCount);* requiring of VC++ and hardware's support and timing is accurate to 1/**lpFrequency** second (e.g. 1/3579545 = 279.36ns in Compaq laptop computer whose CPU is AMD Turion 64 Mobile Technology ML-32 with frequency of 1.75GHz)
5	*int gettimeofday(struct timeval *tv, struct timezone *tz);* timing is accurate to 1μs
6	*int clock_gettime(clockid_t clk_id, struct timespec *tp);* timing is accurate to 1ns
7	*clock_t times(struct tms *buffer);* timing is accurate to 1/_SC_CLK_TCK seconds

Time resolution and upper limit are two main factors that must be considered for selecting a proper time calculation method. Because it is not too long (within 1 minute) for a program to solve a Sudoku puzzle, time resolution is used as the unique reference for method selection. The prototype is running in Windows XP operating system and our computer supports the high time resolution provided by VC++, so the 4th method in Table 1 is selected to computing the time of algorithm execution in this paper. Therefore, the header file <windows.h> has to be included at the beginning of the program.

Firstly, the function of QueryPerformanceFrequency is called and its return value is checked in order to confirm that the computer hardware supports such high time resolution method. And if so, the corresponding frequency is got. Then the function of QueryPerformanceCounter is called before and after the procedure of solving Sudoku puzzle and two count values are obtained respectively. The quotient of their difference and the frequency is just the time that it takes to solve the Sudoku puzzle.

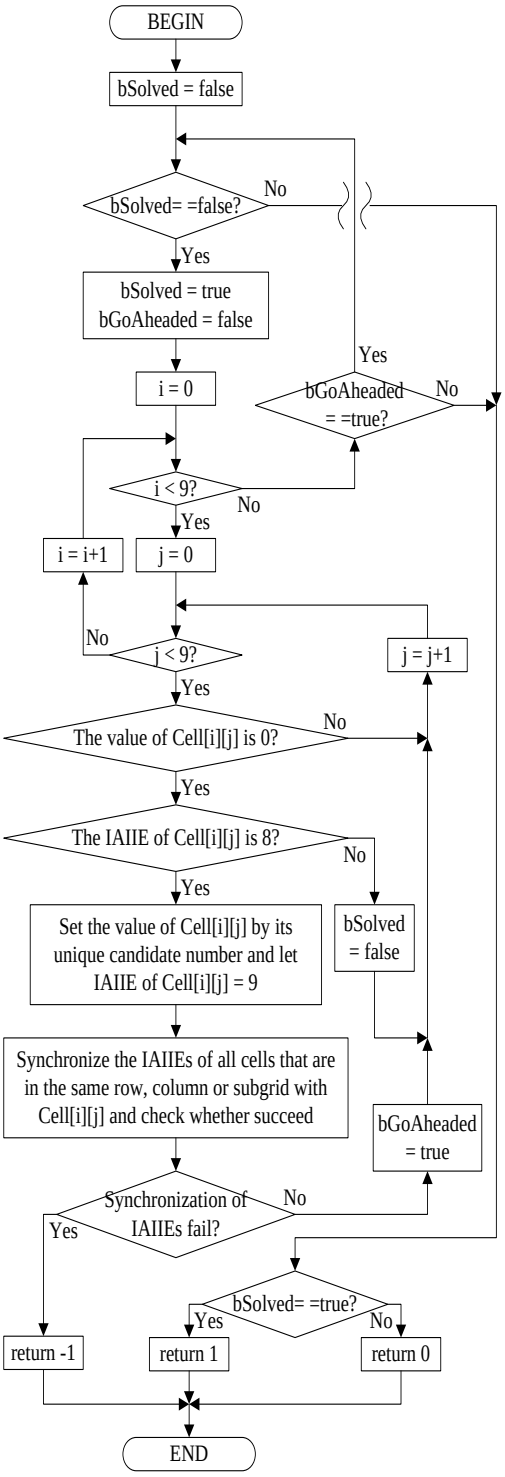

Figure 2. Flowchart of the Straightforward Solving Procedure

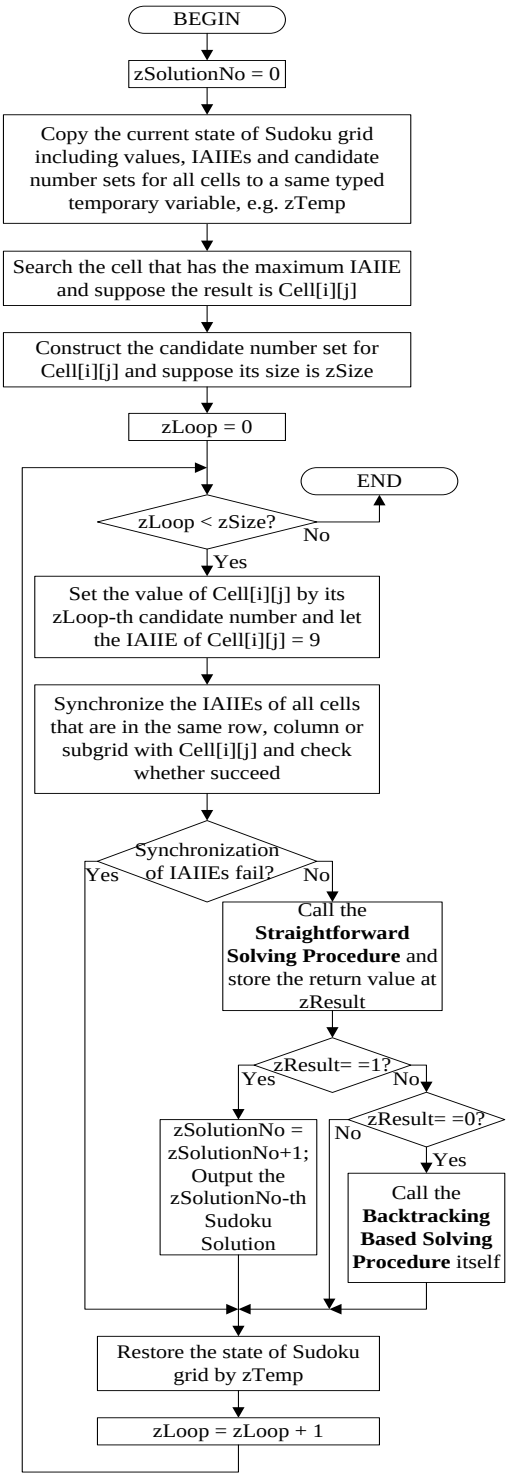

Figure 3. Flowchart of the Backtracking Based Solving Procedure

4. Empirical Studies

4.1. Prototype Implementation

The prototype is implemented in the integrated development environment of Microsoft Visual C++ 6.0 and is running on Windows XP operating system. And the program is developed in standard C++ language using object oriented method.

4.2. Compared with AI Methods

Genetic algorithm is a search heuristic that mimics the process of natural evolution and it is one of the favorite AI methods to be used to solve Sudoku puzzles. Liu Yan-feng and etc. have improved the solving efficiency of genetic algorithm by introducing local searching [5]. Thereafter, their results are selected to be compared with that of the algorithm in this paper (refer to Table 2) as to same Sudoku puzzles whose serial number are all 1 at difficulty levels of easy, medium, hard and evil provided at the Sudoku website www.websudoku.com.

Table 2. Compare the Algorithm Based on Information Entropy and Genetic Algorithm

Difficulty level	Genetic algorithm			Algorithm based on information entropy
	Success rate	Min time	Average time	Time consuming
easy	1.0	7s	3s	3.0515ms
medium	1.0	14s	7s	4.6084ms
hard	0.9	19s	13s	4.6687ms
evil	0.7	27s	15s	5.9337ms

It's obviously that it takes much less time for the algorithm based on information entropy to solve the same Sudoku puzzle. Although the computer hardware configuration in our experiment (*i.e.* 1.75GHz CPU and 896MB memory) is a little better than that in Liu Yan-Feng's experiment (*i.e.* 1.70GHz CPU and 504MB memory). Because time consuming are not at the same order of magnitude, it can be concluded that the algorithm based on information entropy is superior to the genetic algorithm in aspect of time efficiency.

Moreover, there is some success rate problem for the genetic algorithm. In another word, it can't solve hard or evil Sudoku puzzles in some cases. But no Sudoku puzzle is found that can't be solved by the algorithm based on information entropy in this paper till now.

4.3. Compared with Rule based Methods

Stepwise enumerative algorithm based on rule designed by Xiao Hua-yong is an excellent algorithm to solve Sudoku puzzles especially in aspect of time efficiency [2]. And it takes 13ms/416ms to solve the Sudoku puzzle in French finals and the so-called difficult Sudoku puzzle (refer to Figure 1-A and Figure 1-B) respectively on a computer having a 1.86GHz CPU. But it takes only 4.6565ms/158.54026ms to solve the same puzzle for the algorithm in this paper and the running platform (*i.e.* 1.75GHz CPU) is not as good as that for the former.

At the same time, the algorithm in this paper can as well get all proper solutions for those Sudoku puzzles that have multiple solutions and it is verified by using our prototype to solve a Sudoku puzzle with 34 solutions (refer to Figure 1-C) successfully.

5. Discussion about Difficulty Grading Based on Customized Information Entropy

The number of blank cells and the position of blank cells are often used as metrics to define the difficulty levels. And it can be found out that the information entropy of a Sudoku puzzle reflects both the number of blank cells and the position of blank cells. Therefore, it is natural to use the information entropy of a Sudoku puzzle to grade difficulty level of Sudoku puzzles.

40 Sudoku puzzles randomly selected from the Sudoku website www.websudoku.com (10 puzzles at difficulty levels of easy, medium, hard and evil respectively) are solved by the prototype based on the information entropy and corresponding calculation time and information entropy of Sudoku puzzle are recorded. Furthermore, these Sudoku puzzles are sorted according to calculation time and information entropy respectively (refer to Table 3 and Table 4).

Table 3. Sorting Sudoku Puzzles by their Information Entropy

Difficulty Level	Puzzle No.	Information Entropy of Sudoku Puzzle
easy	9,717,341,655	0.15226
easy	4,762,445,254	0.16598
easy	1,004,704,601	0.16735
easy	3,674,910,683	0.17421
easy	2,011,917,968	0.17695
easy	2,221,192,127	0.17833
easy	1,420,320,326	0.18244
easy	1,952,656,679	0.18656
easy	2,537,030,897	0.18656
easy	8,194,651,834	0.18793
medium	3,306,643,642	0.19890
medium	9,049,234,616	0.21125
medium	8,455,601,020	0.21948
medium	1,131,617,241	0.22085
medium	5,509,777,213	0.22085
medium	996,791,631	0.22085
medium	6,548,866,565	0.22634
medium	9,204,260,534	0.23320
hard	8,780,652,748	0.23457
medium	7,420,389,119	0.23594
hard	8,491,219,209	0.23868
medium	2,565,750,554	0.24554
hard	9,948,496,887	0.25103
hard	5,064,052,230	0.25240
hard	6,086,278,933	0.25652
hard	6,515,977,774	0.26063
evil	6,506,868,992	0.26337
hard	4,037,830,336	0.26475
hard	3,582,082,625	0.26749
hard	9,278,338,639	0.27023
evil	5,639,681,400	0.27023
evil	3,296,132,484	0.27298
evil	4,194,163,232	0.27435
evil	279,540,622	0.27984
evil	1,311,588,259	0.27984

evil	6,261,993,052	0.28121
hard	6,647,450,808	0.28258
evil	8,061,665,362	0.28395
evil	9,555,852,040	0.28395
evil	1,990,143,435	0.28807

Table 4. Sorting Sudoku Puzzles by their Solving Time using the Prototype

Difficulty Level	Puzzle No.	Time Consuming (s)
easy	8,194,651,834	0.003015467049583
easy	2,221,192,127	0.003041727370378
easy	3,674,910,683	0.003054857530776
easy	1,004,704,601	0.003055695626120
easy	2,011,917,968	0.003062959119106
easy	4,762,445,254	0.003070222612092
medium	9,049,234,616	0.003091174995705
easy	9,717,341,655	0.003100673409609
medium	1,131,617,241	0.003101232139839
easy	1,952,656,679	0.003108774997940
easy	2,537,030,897	0.003127492460634
medium	5,509,777,213	0.003313828992232
easy	1,420,320,326	0.004517054541848
medium	3,306,643,642	0.004545829148677
evil	6,506,868,992	0.004559518039304
hard	8,780,652,748	0.004577118041539
medium	996,791,631	0.004601143441415
hard	9,278,338,639	0.004635505350540
hard	9,948,496,887	0.004635784715655
medium	6,548,866,565	0.004657295829498
evil	9,555,852,040	0.004694730754886
medium	2,565,750,554	0.004707581550169
medium	8,455,601,020	0.004723505361715
hard	6,086,278,933	0.004751721238314
hard	6,647,450,808	0.004757029175496
hard	4,037,830,336	0.004872406967925
medium	7,420,389,119	0.004889448239930
hard	8,491,219,209	0.004924927609515
hard	6,515,977,774	0.005040305401944
hard	5,064,052,230	0.005196749866254
medium	9,204,260,534	0.005248991142729
hard	3,582,082,625	0.005416051481403
evil	6,261,993,052	0.005483099108965
evil	5,639,681,400	0.005905219797488
evil	1,990,143,435	0.005942934087992
evil	4,194,163,232	0.006016407113195
evil	8,061,665,362	0.006449981771426
evil	1,311,588,259	0.006469816694580
evil	3,296,132,484	0.006504178603705
evil	279,540,622	0.012500750793746

It can be seen from Table 3 that the information entropy of Sudoku puzzles is consistent with their difficulty levels provided by the website on the whole. Although the information entropy of Sudoku puzzle No. 8,780,652,748 at hard level is less that of Sudoku puzzles No. 7,420,389,119 and No. 2,565,750,554 at medium level, the

calculation time of Sudoku puzzle No. 8,780,652,748 is also less than that of some Sudoku puzzles at medium level including puzzles No. 2,565,750,554 and No. 7,420,389,119. Similarly, the information entropy of Sudoku puzzle No. 6,506,868,992 at evil level is less that of Sudoku puzzles No. 4,037,830,336, No. 3,582,082,625 and No. 9,278,338,639 at hard level, but the calculation time of Sudoku puzzle No. 6,506,868,992 is also less than that of all Sudoku puzzles at hard level. Therefore, difficulty grading based on information entropy is rational and it is even more rational than the website in some aspects.

In addition, transformation based on exchange between two rows/columns crossing the same subgrid or between two triple-rows/ triple-columns crossing different subgrids respectively and symmetrical or revolving transformation can be used to normalize and simplify the set of solutions of Sudoku puzzles [3]. And similar transformation is used to generate Sudoku puzzles [8]. At the same time, it is evident that both the number of blank cells and the relative position among value of cells, which are closely related to its difficulty level, keep invariable during such transformation as to a Sudoku puzzle. Accordingly, the following hypothesis can be put forward as an axiom as to difficulty grading method about Sudoku puzzles.

Conjecture 1 *A good difficulty grading method about Sudoku puzzles ought to have such features that the difficulty level of a Sudoku puzzle is always the same as the difficulty level of the Sudoku puzzle resulted from transforming it based on exchange between two rows/columns crossing the same subgrid or between two triple-rows/triple-columns crossing different subgrids respectively, or transforming it symmetrically along horizontal/vertical orientation or along one of its diagonals, or rotating it 90°or 180°clockwise or counter-clockwise around its center cell.*

The above assertion can be called for short as the conjecture of constant difficulty level based on geometric transformation about Sudoku puzzle. Similarly, the following hypothesis can be proved to be tenable as to information entropy of a Sudoku puzzle.

Theorem 1 *The information entropy of a Sudoku puzzle is always the same as the information entropy of the Sudoku puzzle resulted from transforming it based on exchange between two rows/columns crossing the same subgrid or between two triple-rows/triple-columns crossing different subgrids respectively, or transforming it symmetrically along horizontal/vertical orientation or along one of its diagonals, or rotating it 90°or 180°clockwise or counter-clockwise around its center cell.*

An original Sudoku puzzle is given as Fig.4-A so as to prove the above theorem, where X[i,j] represents the value of number filled in corresponding cell[i,j] and i and j represent row number and column number respectively. Note that if cell[i,j] is blank, then the corresponding X[i,j] can be set as 0.

For transformation based on exchange between two rows/columns crossing the same subgrid or between two triple-rows/ triple-columns crossing different subgrids respectively, it is clear that the numbers that are in the same row, column or subgrid with any X[i,j] will keep invariable after transformation. Thus the information entropy of any cell will keep invariable after transformation. This means the information entropy of a Sudoku puzzle will keep invariable after transformation.

For the Sudoku puzzle resulted from transforming the original Sudoku puzzle symmetrically along the vertical axis (consist of such cells as cell[0,4], cell[1,4], cell[2,4], cell[3,4], cell[4,4], cell[5,4], cell[6,4], cell[7,4], cell[8,4], refer to Fig.4-B) or along the horizontal axis (consist of such cells as cell[4,0], cell[4,1], cell[4,2], cell[4,3],

cell[4,4], cell[4,5], cell[4,6], cell[4,7], cell[4,8]), it is also true that the numbers that are in the same row, column or subgrid with any X[i,j] will keep invariable after transformation. So the information entropy of a Sudoku puzzle will keep invariable after such horizontal or vertical symmetric transformation.

For the Sudoku puzzle resulted from transforming the original Sudoku puzzle symmetrically along the diagonal from top right corner to bottom left corner (consist of such cells as cell[0,8], cell[1,7], cell[2,6], cell[3,5], cell[4,4], cell[5,3], cell[6,2], cell[7,1], cell[8,0], refer to Fig.4-C) or along the diagonal from top left corner to bottom right corner (consist of such cells as cell[0,0], cell[1,1], cell[2,2], cell[3,3], cell[4,4], cell[5,5], cell[6,6], cell[7,7], cell[8,8]), or from rotating the original Sudoku puzzle 90° clockwise (refer to Figure 4-D)/ counter-clockwise (equivalent to 270° counter-clockwise/clockwise respectively) around its center cell (*i.e.* the cell of cell[4,4]), it can be seen that the numbers that are in the same subgrid with any X[i,j] will also keep invariable after transformation, but the numbers that are in the same row with X[i,j] become the numbers that are in the same column and vice versa. This means that the information entropy of any cell will also keep invariable after transformation. So the information entropy of a Sudoku puzzle will keep invariable after such diagonal based symmetric transformation or rotating 90° clockwise/ counter-clockwise based transformation.

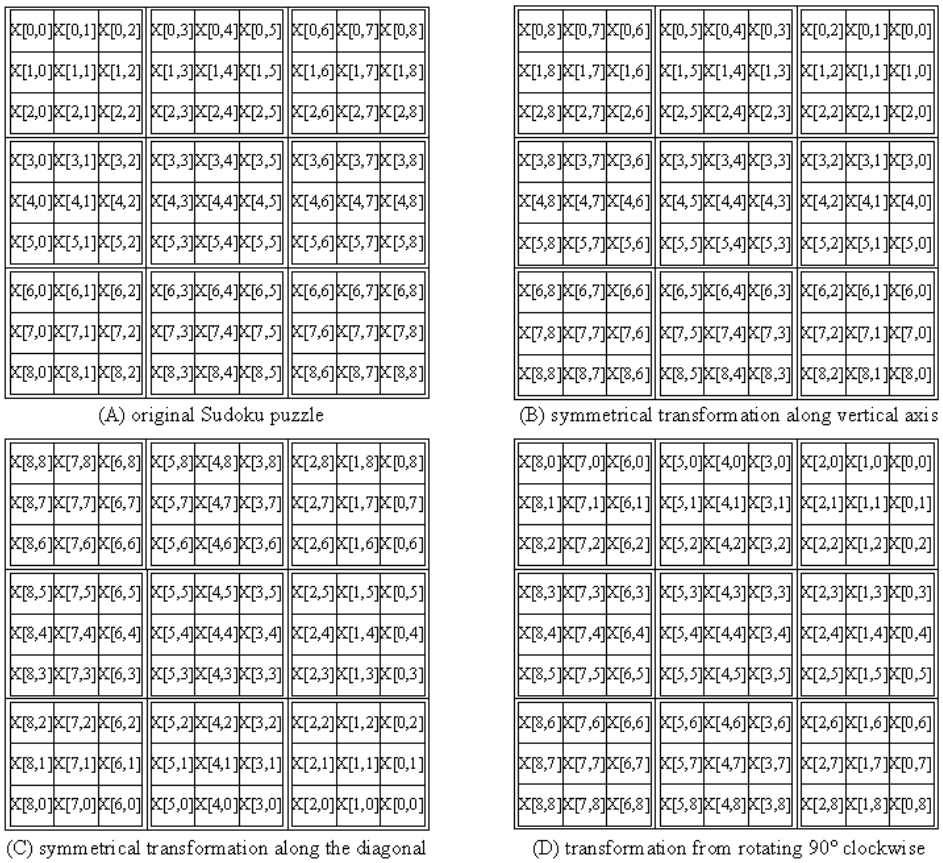

Figure 4. Original Sudoku Puzzle and its Geometric Transformation

For the Sudoku puzzle transformed from rotating the original Sudoku puzzle 180° clockwise (equivalent to 180° counter-clockwise) around its center cell, it can also be concluded that the numbers that are in the same row, column or subgrid with any X[i,j] will also keep invariable after transformation. So the information entropy of a Sudoku puzzle will keep invariable after such rotating 180° clockwise based transformation.

To sum up, it is suitable to grade difficulty of Sudoku puzzles according their information entropy and it is convenient for the corresponding method satisfy the condition of constant difficulty level based on geometric transformation.

Moreover, fuzzy clustering methods can be used to accomplish difficulty grading of Sudoku puzzles according to their information entropy.

6. Summary

Experimental results show that the algorithm based on customized information entropy has better time efficiency than available methods including generic algorithms and rule based algorithms. Meanwhile, the algorithm has the capability of solve difficult puzzles that available software can't solve and it can solve not only unique-solution puzzles but also multiple-solution puzzles. In addition, it is potentially a feasible choice to grade difficulty of Sudoku puzzles based on information entropy.

Acknowledgements

This research was supported in part by the Fundamental Research Funds for the Central Universities (No.2009JBM019).

References

[1] J. Delahaye, "The science behind Sudoku", Scientific American, vol. 294, no. 6, **(2006)**, pp. 80-87.
[2] H. Xiao, Z. Tian and L. Ma, "Design of stepwise enumerative algorithm based on rule about Sudoku", Computer Engineering and Design, vol. 31, no. 5, **(2010)**, pp. 1035-1037, 1113.
[3] Z. Zhang, "Solve & generate Sudoku puzzle by programming in AutoCAD", Computer Programming Skills & Maintenance, no. 17, **(2008)**, pp. 16-18, 21.
[4] J. Xu, "Using backtracking method to solve Sudoku puzzle", Computer Programming Skills & Maintenance, no. 5, **(2009)**, pp. 17-21.
[5] Y. Liu and S. Liu, "Algorithm based on genetic algorithm for Sudoku puzzles", Computer Science. vol. 37, no. 3, **(2010)**, pp. 225-226, 233.
[6] T. Mantere and J. Koljonen, "Solving, rating and generating Sudoku puzzles with GA", In: Proceedings of 2007 IEEE Congress on Evolutionary Computation (CEC 2007), **(2007)**, pp. 1382-1389, IEEE, New York.
[7] T. Mantere and J. Koljonen, "Solving and analyzing Sudokus with cultural algorithms", In: Proceedings of 2008 IEEE Congress on Evolutionary Computation (CEC 2008), **(2008)**, pp. 4053-4060, IEEE, New York.
[8] Z. Zhao, J. Guo and L. Yang, "Study of algorithm about Sudoku generation", Neijiang Science and Technology, no. 7, **(2008)**, pp. 22-23, 143.

Authors

Gaoshou Zhai received the B.Sc., Master and Ph.D. degrees in 1993, 1996 and 2000 respectively. From 2000 to 2002, he was a lecturer in the School of Computer and Information Technology at Beijing Jiaotong University. Since 2002 he has been an associate professor and since 2007 he has been vice director of the Department of Computer Science. From January to May in 2006, he had been to the department of computer science at UIUC as a visiting scholar. His research interests include operating systems, system security, system software design and automatic tools for software engineering, algorithm analysis and design, artificial intelligence and intelligent traffic systems. In these areas he has published more than 40 papers in journals and conference proceedings. He served as program committee member of SERA2009 and invited review specialist for Chinese Science and Technology Papers Online sponsored by Centre for Science and Technology Development, MEPRC. He is also invited to review papers for Journal of Xi'an Jiaotong University, Journal of Beijing Jiaotong University, Journal of Lanzhou Jiaotong University, ICCIT2009 and ICSAI2012. He is a member of ACM and SERSC, a senior member of CCF and IACSIT.

Jun-Hong Zhang received the B.E. degree in measurement and control technology and instrumentation and M.E. degree in safety technology and engineering, from Beijing Jiaotong University, China, in 1998 and 2003 respectively. Now she is a lecturer at the School of Electrical and Information Engineering, Beijing University of Civil Engineering and Architecture while she is also an on-the-job doctoral student in the major of traffic information and control engineering at School of Electronics and Information Engineering, Beijing Jiaotong University. Her major research interests include computer simulation, traffic information and control, algorithm analysis. In these areas she has published 6 papers in journals and conference proceedings.