International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:8, No:10, 2014

Rating and Generating Sudoku Puzzles Based On Constraint Satisfaction Problems

Bahare Fatemi, Seyed Mehran Kazemi, Nazanin Mehrasa

***Abstract*—**Sudoku is a logic-based combinatorial puzzle game which people in different ages enjoy playing it. The challenging and addictive nature of this game has made it a ubiquitous game. Most magazines, newspapers, puzzle books, etc. publish lots of Sudoku puzzles every day. These puzzles often come in different levels of difficulty so that all people, from beginner to expert, can play the game and enjoy it. Generating puzzles with different levels of difficulty is a major concern of Sudoku designers. There are several works in the literature which propose ways of generating puzzles having a desirable level of difficulty. In this paper, we propose a method based on constraint satisfaction problems to evaluate the difficulty of the Sudoku puzzles. Then we propose a hill climbing method to generate puzzles with different levels of difficulty. Whereas other methods are usually capable of generating puzzles with only few number of difficulty levels, our method can be used to generate puzzles with arbitrary number of different difficulty levels. We test our method by generating puzzles with different levels of difficulty and having a group of 15 people solve all the puzzles and recording the time they spend for each puzzle.

***Keywords*—**Constraint satisfaction problem, generating Sudoku puzzles, hill climbing.

I. Introduction

SUDOKU is a logical game which has attracted both young and old people. Having a very challenging and addictive nature, it has spread wildly all over the world. The word Sudoku is short for *Su-ji wa dokushin ni kagiru* which means "the numbers must be single". This name indicates the nature of the game, in which numbers should be placed in appropriate places in a grid.

The first Sudoku was published in a puzzle magazine in USA, 1979 [1]. The objective of the game is to take a 9×9 grid and fill in the open spots with numbers from 1 to 9 so that each column and each row of the grid contains each of the numbers only once. Furthermore, each of the nine 3×3 sub-grids that together compose the total 9×9 grid (also called *boxes*, *blocks*, *regions*, and *sub-squares*) must contain all of the digits from 1 to 9 only once.

A Sudoku has at least 17 pre-defined numbers but normally there are 22 to 30. The difficulty level is mainly determined by the amount of empty cells. For instance, in a beginner puzzle several numbers will be given. In more advanced puzzles, only a few numbers are given. But this is not the only factor for determining the level of difficulty. There are also other factors such as the lowest bound of the given cells in each row and column and applicable techniques by human logic thinking which can affect the difficulty level of a Sudoku puzzle.

Fig. 1 is an example of a Sudoku puzzle. We can see in this figure that 23 of the cells are initially containing a number. The aim of the game is to fill the other cells so that they don't violate the conditions mentioned earlier. Fig. 2 represents the solution to the Sudoku puzzle appearing in Fig. 1. We can see in final solution that every row, column and sub-square contains all the numbers from one to nine only once.

Several magazines, newspapers, puzzle books, etc. publish a great number of Sudoku puzzles with different levels of difficulty every day. The difficulty level of the puzzles can help people in choosing a puzzle according to their level of knowledge and skill in solving Sudoku. This ubiquity of the game has raised the problem of generating lots of puzzles with different levels of difficulty.

In this paper, we will examine the problem of generating Sudoku puzzles with different levels of difficulty. We introduce constraint satisfaction problems and then formulate each puzzle as a constraint satisfaction problem and try to solve it using the arc consistency with domain splitting. We use the number of calls to the arc consistency function for rating the difficulty level of the puzzles. We also introduce hill climbing method and propose an algorithm based on it to generate puzzles with different levels of difficulty. We test the performance of our algorithm in generating puzzles with different levels of difficulty by asking a group of people to solve our puzzles and recording the time they spent on each puzzle.

The rest of the paper is organized as follows: Section II is a literature review of different methods of generating Sudoku puzzles in the literature. Section III provides sufficient background for readers to go through the detail of the paper. Section IV of the paper represents how we formulate each Sudoku puzzle as a constraint satisfaction problem, how we solve it, and how we rate the difficulty level of puzzles. In Section V, we explain our hill climbing method for generating puzzles with different levels of difficulty. Section VI is devoted to evaluation of our method. Finally, Section VII concludes the paper and points out some future works.

Bahare Fatemi and Nazanin Mehrasa are with the Computer engineering and IT Department, Amirkabir University of Technology, Iran (e-mail: b.fatemi@aut.ac.ir, nazanin.mehrasa@aut.ac.ir)

Seyed Mehran Kazemi is with the Computer Science Department, University of British Columbia (e-mail: smkazemi@cs.ubc.ca).

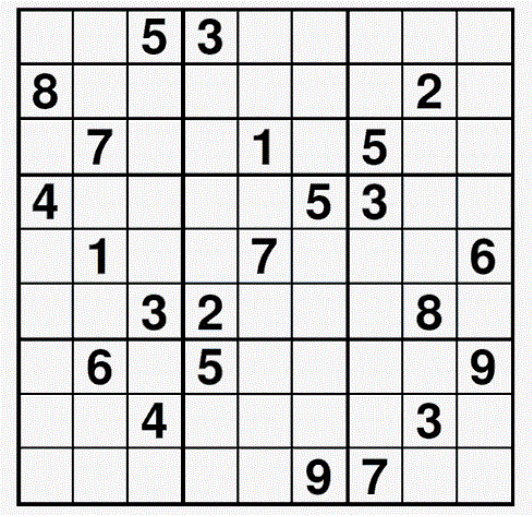

Fig. 1 An example of a Sudoku puzzle

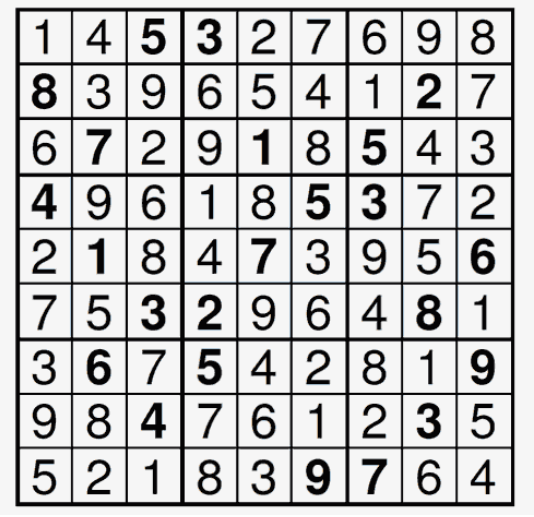

Fig. 2 The solution to the Sudoku puzzle in Fig. 1

II. Literature Review

There are several methods in the literature proposed for generating Sudoku puzzles with different levels of difficulty. Most of these methods rely on an algorithm for solving Sudoku puzzle. Solving the generalized Sudoku problem is NP-complete, as has been shown in [2]. Therefore, we cannot hope to find an algorithm with polynomial time for all puzzles, unless P = NP [3]. This means that there will be possibly many instances that cannot be solved without one kind of search also being necessary [4]. Consequently, Sudoku designers try to propose an algorithm based on stochastic local search or other optimization methods for solving the Sudoku puzzles and they consider the time spent by this algorithm for solving a given puzzle to be indicative of its difficulty level. For instance, one of these methods [5] solves puzzles using genetic algorithms, a computer-based optimization method which uses the Darwinian evolution of nature as an inspiration and models the world accordingly. In this method, a genetic algorithm formulation is proposed to solve Sudoku puzzles. Since different puzzles need different amount of time and different number of generations to be solved using this method, these criteria have been considered as the measure for difficulty level for each Sudoku. The genetic algorithm proposed in [5] for solving Sudoku puzzles has been improved later by Kazemi and Fatemi [6], offering a possibly better alternative for generating puzzles with different levels of difficulty using genetic algorithm.

The work in [7] is another example which frames the Sudoku puzzle as a search problem and uses the expected search time to determine the difficulty level of each puzzle.

Another method [8] for generating Sudoku puzzles deals with the problem as an inverse problem. It starts with a completed Sudoku board and applies inverse methods to construct a puzzle with a small set of pre-defined cells, such that it has only a unique solution.

Using dig-hole strategy on a valid grid [9] is another famous method for generating Sudoku puzzles. There are two major steps in this method. The first step is to create a valid grid using the Las Vegas algorithm and the second step is erasing some of the digits by using special operations. (To see other methods for generating Sudoku puzzles see [10]).

Most of previous methods for generating Sudoku puzzles are designed to generate puzzles with only a few different levels of difficulty.

In this paper, we try to propose a different way of generating Sudoku puzzles, in which we address the problem of generating puzzles with manifold different levels of difficulty. First of all, we propose a way of measuring the difficulty level of each puzzle by formulating it as a Constraint Satisfaction Problem (CSP) and solving it by arc consistency and domain splitting. Using this method, we can categorize puzzles into our desirable number of difficulty levels. Then we propose a hill climbing algorithm for generating puzzles. Given the desired level of difficulty, this method can produce a puzzle in that level of difficulty. We test our model by generate four puzzles with four levels of difficulty and have a group of 15 people solve all four puzzles. These people have different ages and different academic backgrounds, but all of them have solved plenty of Sudoku puzzles before. They also have different strategies for solving a puzzle. We record the time they spend for solving each Sudoku puzzle to evaluate our method.

III. Background

In this section, we provide sufficient information about constraint satisfaction problems and hill climbing which a reader needs to know to read the rest of the paper.

A. Constraint Satisfaction Problem

A Constraint Satisfaction Problem (CSP) [11] is defined as a set of variables, the domain of possible values for each variable, and a set of constraints between one or more

variables. The solution to a CSP is an assignment of values to all variables from their domain which satisfying all constraints.

There are several methods proposed for solving CSPs. The method which we will use in this paper is based on arc consistency and domain splitting. Consistency techniques try to reduce the search space by removing the values that cannot appear in a solution. Domain splitting tries to decompose the problem into smaller problems and searches within each sub-problem to find a solution.

To solve a CSP using arc consistency with domain splitting, first of all a constraint network is generated. For building a constraint network, we draw an oval for each variable and a rectangle for each constraint. There are arcs (undirected edges) between variable nodes and constraint nodes whenever a given variable is involved in a given constraint.

Then we try to make each arc consistent. An arc between a variable *x* and a constraint *r(x, y)* is consistent if for each value *x** in domain of *x,* there is a value *y** in domain of y so that *r(x*, y*)* is satisfied. A network is arc consistent if all its arcs are arc consistent. We can make an arc consistent by removing those values from the domain of its associated variables which violate the consistency.

When the network is arc consistent, if some of the variables have more than one value left in their domain, we choose one of these variables randomly and split its domain into two or more disjoint groups. This is called domain splitting. Then we have a number of smaller problems. We apply arc consistency and domain splitting to each of them until we find a solution.

B. Hill Climbing Algorithm

Hill climbing [12] is a mathematical optimization technique which belongs to the family of local search. It is used to minimize the cost function or maximize the utility function given for a problem. It is an iterative algorithm that starts with an arbitrary solution to a problem and calculates the cost (utility) function for the given solution. Then at each iteration, it changes one element of the solution and calculates the cost (utility) function again. If the change resulted in a decrease (increase) in the cost (utility) function, the change is accepted and otherwise it is undo-ed. This process continues until a desirable solution to the problem is found or until it times out.

IV. Rating Sudoku puzzles

In order to rate the difficulty level of each Sudoku puzzle, we formulate it as a CSP and solve it by arc consistency with domain splitting. We count the number of times arc consistency function is called by each puzzle and use it as the measure of difficulty. In the rest of the paper, we call this count for each puzzle *call count* of the puzzle. Our aim is to determine the value of call count for different levels of difficulty and generate puzzles having a call count close to values we determined for each level. To determine the appropriate values of call counts for each level of difficulty, we solve several puzzles in each level, record the number of calls to the arc consistency function and take the average.

In modeling Sudoku as a CSP, we need to specify the variables, domains of the variables and constraints between variables. In our model of CSP, variables are the cells of the Sudoku puzzle. We represent the set of variables as $\{x_{11}, x_{12},\ldots, x_{19}, x_{21}, \ldots, x_{99}\}$. The domain of variables for having no pre-defined values is {1,2,3,4,5,6,7,8,9}.

This means that a cell without a pre-defined value can take all the values between 1 and 9 inclusive. For cells having a pre-defined value, the domain is just their initial value. For example if the puzzle designer has set the initial value of the cell in the second row and third column to *5*, then the domain of $x_{23}$is *{5}*.

There are three constraints for each of the cells. The first constraint is that no cells in the same row can have the same value. The second one is that no cells in the same column can have the same value. Finally, the third constraint is that no cells in the same sub-square can have the same value. Then we use arc consistency with domain splitting to solve this consistency network. Parts of the consistency network can be seen in Fig. 3.

Arc consistency is applied to the consistency network in Fig. 3 by choosing the arcs which are not consistent and removing the values from its associated variable until the arc is consistent. This process continues until no non-consistent arc consists in the network.

In order to apply domain splitting to a consistent network, we randomly choose a variable x_{ij} from the network which has $k > 1$ values in its domain. Then we split the current node into *k* nodes each having only one of the values in the domain of x_{ij}. After that we run the arc consistency for each of these nodes. This tree of nodes is generated using depth first search. Splitting of nodes in a branch continues until we reach a node in which all variables have only one variable in their domain or one of the variables has a value in its domain. The former indicates a solution is found and the latter indicates no solution can be found.

A. Data Collection

In order to determine the number of splits required to solve puzzles with different levels of difficulty, we collected 400 Sudoku puzzles from *www.websudoku.com* where 100 of them were easy, 100 of them were medium, 100 of them were hard and 100 of them were evil. Then for each of these puzzles, we ran our algorithm 10 times and took the average call count of the 10 runs for solving each puzzle. We observed some outliers among each set of 100 call counts for each difficulty level. This can be due to misclassification of puzzles by the website or because of the reason that they included some of the puzzles in more than one difficulty level. We removed the outliers using the Inter-Quartile Range (IQR) method.

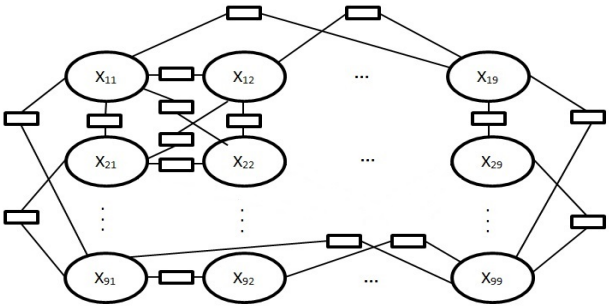

Fig. 3 Parts of the consistency network used in our CSP formulation of Sudoku puzzles.

In this method, we calculate the IQR of the data as the difference between the third and the first quartiles of the data. First (third) quartile of a set of data points is the number which 25% (75%) of the data are less than that. Values beyond [Median - 1.5 * IQR, Median + 1.5 * IQR] were considered as outliers and removed. Median in this interval indicates the number which 50% of the data are less than that. The distribution of the data over the call count (after outlier removal) is presented in Fig. 4. Finally, we calculated the average of the call counts for the remaining data. These average numbers of every level represent the average call count of each difficulty level. These numbers are presented in Table I.

V. Generating Sudoku puzzles

The results of the previous section can help us generate Sudoku puzzles with different levels of difficulty. As we can see in Table I, different levels of difficulty have different values of call count. The harder the puzzle, the more its call count. We can use this fact to generate Sudoku puzzles with arbitrary levels of difficulty. We only need a mechanism to generate puzzles with call counts close to the call count value that we desire.

Suppose we want to generate puzzles with 4 levels of difficulty. We can consider the values in Table I as the call counts for these levels and then try to generate tables having call counts close to these numbers. If we need more than 4 difficulty levels, we can consider the mean of each two consecutive number in Table I as a new call count for a difficulty level. We can also consider the numbers beyond the last number in Table I. This enables us to generate puzzles with arbitrary different levels of difficulty.

TABLE I
Average Call Counts of Four Difficulty Levels

Difficulty	*Easy*	*Medium*	*Hard*	*Evil*
Average	6.234043	29.2093	98.2093	527.4318

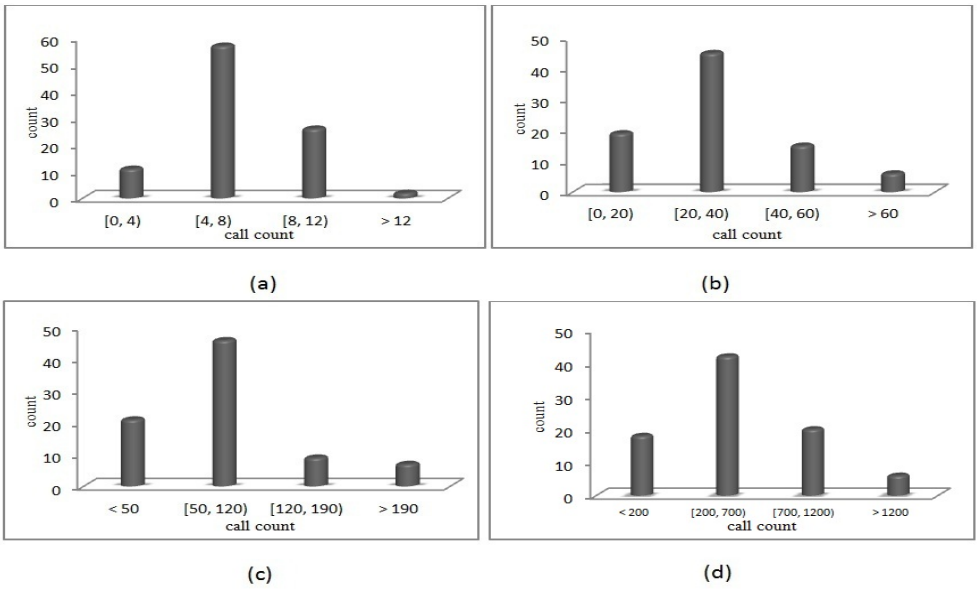

Fig. 4 The distribution of the call counts of the puzzles having a difficulty level of (a) easy, (b) medium, (c) hard, and (d) evil

We use hill climbing to generate new puzzles having a call count close to the call count we need. In this method, first of all, we generate an initial puzzle with some random numbers inside it and calculate its cost function.

Then in each iteration, we randomly change one element of this solution by adding, deleting, or changing a single number and calculate the cost function again. After this, we check the new value of the cost function for this new puzzle and compare it to the previous one. If the cost is reduced, we accept the second puzzle as the new solution and otherwise we undo the change. We do this process until meeting the stopping criterion.

The cost function for a given puzzle is infinity for puzzles with none or more than one solutions. For other puzzles, the cost is the absolute value of the current puzzle's call count minus the average call count of the given difficulty level. For

example if we want to generate an easy puzzle and we want to consider the values demonstrated in Table I, then the cost function for a puzzle having a unique solution is the absolute value of its call count minus 6.234043.

We stop the algorithm when we have puzzles with costs close to zero. Depending on the level of accuracy we need and the number of difficulty levels we have, we can define the closeness of the cost function to zero.

VI. Evaluation

In order to test our method we generated 4 puzzles with 4 different levels of difficulty. We used the call counts in Table I as the average call counts of our puzzles. Then, we had agroup of 15people having different ages and different levels of education solve all four puzzles and we measured the amount of time they spent for each of the puzzles. The age of the people in this group was between 16 and 41 and they had different levels of education from high school students to graduate students and also non-student people from workers to people having occupations requiring great skills of problem solving. All these people had solved plenty of Sudoku puzzles before and they were using different strategies for solving a puzzle.

Results obtained from this study are demonstrated in Table II. Each of the rows in this table represents the time in minutes they spent for solving each of the puzzles having different levels of difficulty. The last row represents the average time in minutes that people spent on solving each of the puzzles.

We can see from the results in Table II that most people spent more times for solving more difficult puzzles. There are some exceptions where people did a better job for more difficult puzzles which can be because of the strategy they follow. It is possible that their strategy works better for a more difficult puzzle than an easier puzzle. We can also see from average times in the last column that people, on average, spent more time on more difficult puzzles. This indicates that our method has succeeded in generating puzzles with different levels of difficulty.

TABLE II
Times Spent by People for Solving Our Puzzles

	Easy	*Medium*	*Hard*	*Evil*
1	16	12	25	30
2	21	36	67	50
3	14	30	80	68
4	15	24	30	68
5	8	22	26	55
6	10	33	24	38
7	22	6	23	35
8	11	20	20	45
9	6	8	12	21
10	6	8	9	12
11	40	45	unsuccessful	unsuccessful
12	8	31	61	71
13	10	9	18	50
14	14	20	25	unsuccessful
15	20	18	30	unsuccessful
Average	14.73	21.47	32.14	45.25

VII. Conclusion

Sudoku puzzle is a popular game among all people having different ages, occupations, levels of education, etc. and is being played by many people every day. Lots of magazines, newspapers, puzzle books, etc. publish Sudoku puzzles with different levels of difficulty for interested people. Designing such a huge number of Sudoku puzzles cannot be performed by human labor. Computer programs are required to do this task. Several methods have been proposed in the literature to generate puzzles with different levels of difficulty.

In this paper, we proposed a method which formulated a Sudoku puzzle as a constraint satisfaction problem and solved it by arc consistency and domain splitting. Then we used the number of calls to the arc consistency function as the difficulty level of the puzzle. We used 400 different puzzles having 4 different levels of difficulty (100 in each level) to determine how many calls to this function are required on average to solve a puzzle in a given level of difficulty. We observed that as the difficulty level of a puzzle increases, the number of calls to the arc consistency function increases accordingly. We called the number of calls to the arc consistency function for a puzzle "call count" of the puzzle.

Then we used a hill climbing method which started with an arbitrary puzzle and tried to add, delete or change its numbers until getting a puzzle which had a call count close to the call count we desires to get for the given difficulty level.

Using our method, we could generate puzzles with many different levels of difficulty. In order to test our method, we generated four puzzles with four different levels and had a group of 15 people solve our puzzles. We recorded the time it took for every person to solve each puzzle. We observed that as the difficulty level of the puzzle generated by our method increased, the average time people spent for solving it also increased.

In future, we can generate more puzzles with more levels of difficulty and ask people to solve our puzzles to test the power of our method in generating more levels of difficulty. Furthermore, we can use a stochastic local search other than hill climbing which is more efficient and can generate a desirable puzzle in a shorter amount of time.

References

[1] R. Lewis, "Metaheuristics can solve sudoku puzzles," Journal of heuristics, vol. 13, no. 4, pp. 387–401, 2007.
[2] H. Poor, *An Introduction to Signal Detection and Estimation*. New York: Springer-Verlag, 1985, ch. 4.
[3] Y. Takayuki and S. Takahiro, "Complexity and completeness of finding another solution and its application to puzzles," IEICE transactions on fundamentals of electronics, communications and computer sciences, vol. 86, no. 5, pp. 1052–1060, 2003.

[4] M. R. Garey and D. S. Johnson, Computers and intractability, vol. 174. Freeman New York, 1979.
[5] T. Mantere and J. Koljonen, "Solving and rating Sudoku puzzles with genetic algorithms," in New Developments in Artificial Intelligence and the Semantic Web, Proceedings of the 12th Finnish Artificial Intelligence Conference STeP, pp. 86–92, 2006.
[6] S. M. Kazemi, B. Fatemi, "A retrievable genetic algorithm for efficient solving of Sudoku puzzles," International Journal of Computer, Information Science and Engineering, Vol. 8, No. 5, 2014.
[7] K. N. Das, S. Bhatia, S. Puri, and K. Deep, "A retrievable GA for solving Sudoku puzzles," Citeseer, 2012.
[8] T. Boothby, L. Svec, and T. Zhang, "Generating sudoku puzzles as an inverse problem," Mathematical contest in modeling, 2008.
[9] C. Chang, Z. Fan, and Y. Sun, "A Difficulty Metric and Puzzle Generator for Sudoku," UMAPJournal, p. 305, 2007.
[10] Y. Xue, B. Jiang, Y. Li, G. Yan, and H. Sun, "Sudoku puzzles generating: from easy to evil," Mathematics in practice and theory, vol. 21, p. 000, 2009.
[11] D. L. Poole and A. K. Mackworth, Artificial Intelligence: foundations of computational agents. Cambridge University Press, 2010.
[12] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards, Artificial intelligence: a modern approach, vol. 74. Prentice hall Englewood Cliffs, 1995.