World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:8, No:8, 2014

A Block World Problem Based Sudoku Solver

Luciana Abednego, Cecilia Nugraheni

Abstract—There are many approaches proposed for solving Sudoku puzzles. One of them is by modelling the puzzles as block world problems. There have been three model for Sudoku solvers based on this approach. Each model expresses Sudoku solver as a parameterized multi agent systems. In this work, we propose a new model which is an improvement over the existing models. This paper presents the development of a Sudoku solver that implements all the proposed models. Some experiments have been conducted to determine the performance of each model.

Keywords—Sudoku puzzle, Sudoku solver, block world problem, parameterized multi agent systems.

I. Introduction

SUDOKU is a very popular puzzle game. The components of this puzzle are a board, which is a 9×9 cells, divided into nine 3×3 subblocks, and a set of numbers from 1 to 9. The objective of this puzzle is to fill in every cell of the board with a number from 1 to 9 such that every row, every column, and every sub-block contains each number exactly one. Fig. 1 is an example of Sudoku Puzzles and its solution.

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

(a)

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

(b)

Fig. 1. An example of Sudoku puzzle and its solution.

There are many approaches for solving Sudoku puzzles, such as integer programming [1], SAT [5], genetic algorithm [6], [2], simulated annealing [7], meta-heuristics [4], neural networks [8], [13], particle swarm optimization [9], [10], and many more. In [11] we proposed an approach for solving Sudoku which is by modeling the puzzles as *block-world problems*. A block-world problem consists of a number of boxes on the table with a particular arrangement and two robots. Initially, all the boxes are on the table and are arranged into a number of piles. The objective of this problem is to change this arrangement into a targeted arrangement. The robots are responsible for changing the arrangement. Each robot has a special capability. The first robot is only capable to take a box from a table and put it on another box, whereas the second robot is capable to take a box from a top of a pile and put it on the table. It is assumed that every time only one robot that can make a move. An example of block world problem is given in Fig. 2.

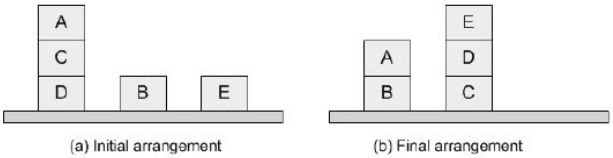

Fig. 2. An example of block world problem.

By modifying some settings of block world problem we have shown that Sudoku puzzles can be regarded as an variant of block world problems. We presented three block world problem based Sudoku solver models. The first model is based on *backtracking* principle. The solution searching is done exhaustively over the problem state space. The searching process stops when a solution is found or when there is no more alternative solution can be found. The second and the third model are based on *fixed point* principle. The solution search process is divided into nine sub-processes $P = \{p_1, p_2, \ldots, p_9\}$. Each sub-process, p_i, tries to seek an empty cell for putting number i regarding some certain conditions/rules. If there are no more numbers can be placed on the board, the searching process halts.

In [11] we have made a manually analysis of the models' performance. By using a Sudoku puzzle, we run each model's algorithm to determine its performance. In this work, continuing our previous work, we defined a new model and developed a Sudoku solver based on the models. We then use the solver to do some experiments. The objective of the experiments are to measure the performance of each model.

The remainder of this paper is organized as follows. In Section II we review our proposed models. Section III explains briefly the implementation principle of Sudoku Solver. Section IV discusses the experimental results. Conclusions and future work are given in Section V.

II. Sudoku solver models

For the sake of clarification, in this section we give a summary of the Sudoku solver modeling presented in [11]. The readers may consult [11] for more detailed explanation.

Following [4], we define a Sudoku puzzle as follows :

Definition 1 Given an $n^2 \times n^2$ cells divided into $n \times n$ distinct subblocks, the aim of Sudoku puzzle is to fill each cell so that the following three criteria are met:

Luciana Abednego and Cecilia Nugraheni are with the Informatics Department, Parahyangan Catholic University, Bandung, Indonesia (e-mail: luciana,cheni@unpar.ac.id).

1) Each row of cells contains the integers 1 through to n^2 exactly once.
2) Each column of cells contains the integers 1 through to n^2 exactly once.
3) Each subblock contains the integers 1 through to n^2 exactly once.

In our work, we focus on $n = 3$.

In the introduction we describe the components of a block world problem and the objective of the problem. Now let's do some modification as follows:

- The number of boxes on the table is 9×9.
- Each box has number on it, which is between 1 and 9.
- For every number, from 1 to 9, there are 9 boxes with each number.
- There is a special part of the table called board which consists of 3×3 grids. Each grid is called subblock and consists of 3×3 smaller grids called cells.
- Initially some of the boxes are already placed on those cells and the rest are outside the board. The boxes that are outside the board are organized in nine piles according to their numbers.

The result of this modication is illustrated in Fig. 3.

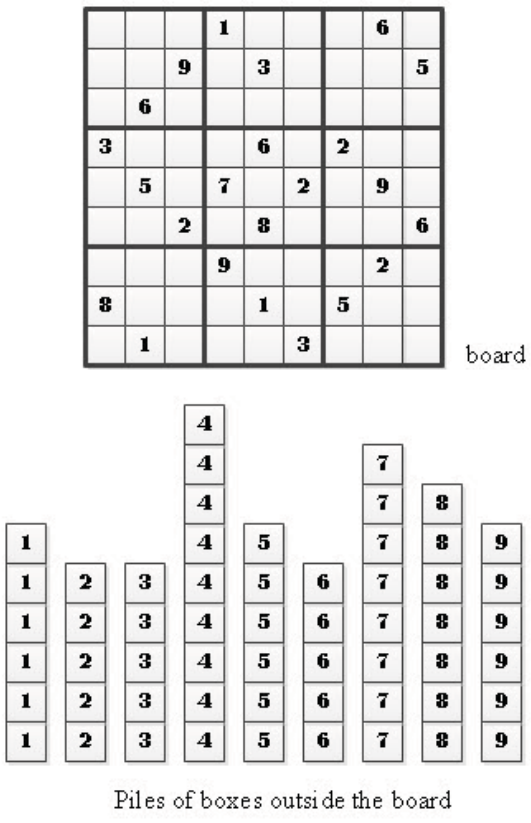

Fig. 3. Modification of a block world problem.

Furthermore, let's make some changes over the robots' behaviours, as follows:

- The first robot is capable of taking a box that are outside the board and putting this box on a cell of the board so that the condition in Definition 1 are satisfied.
- The second robot is capable of taking a box from the board and putting it back on the pile according to its number.

With those changes, the block world problem has became a Sudoku puzzle.

Now, we will briefly present our proposed Sudoku solver models. The models differ in terms of the number of robots and the task of each robot.

A. Model 1

Similar to the original block world problem, the first model uses two robots with different task. These robots will work in a turn based style with the following rules:

- The first robot gets the first turn. The task of this robot is to fill an empty cell with a box from a pile outside the board without violating Sudoku rules. The first robot will do it repeatedly, until one of two conditions is reached:
  - All the cells are filled with boxes (which means no more boxes outside the board). At this situation the first robot will report that the job is done.
  - It is unable to find a box that can be placed on an empty cell. If this happens, then it will report that there is a failure and give the turn to the second robot.

  For the process of searching an empty cell, the first robot will take the empty cell with the lowest position. We define the the lowest and highest position as $(1, 1)$ and $(9, 9)$, respectively. The first robot records all the empty and not empty cells from the beginning until the end of the process.
- The second robot will be active whenever a failure happens. It is responsible for taking a box from the board and put it back on its correspoding pile.
  In our model, the second robot will take the last box put by the first robot. As consequence, the second robot needs information about the order of the box put on the board.

B. Model 2

Differ from the first model, for the other models, we use nine robots. All robots have the same task, which is to take a box from the piles outside the board and put the box on the board's cell.

We use the *fixed-point* principle for this model. The searching for the solution is done by making iterations until a termination condition is reached. In each iteration, every robot i tries to find an appropriate cell or a valid position for a box with number i. If the searching is success, then robot i puts a box with number i on it. After doing its job, the robot i will give the turn to the next robot. After the robot 9 does its job, it will be decided whether a new iteration should be made or not. If in the last iteration, all the robots cannot find appropriate cells, in other word, the robots can not put any boxes on the board anymore, then the process is terminated. In doing its task, a robot i first evaluate an empty cell of the board, one by one from the lowest position to the highest position.

The difference between model 2 and model 3 is on the definition of a valid position. For model 2, we use the following definition for a valid position.

Definition 2 A robot i will call a position (x, y) is valid for box with number i if the following conditions hold:

1) It is empty.
2) The row x does not contain any boxes with number i.
3) The column y does not contain any boxes with number i.
4) The subblock where (x, y) is located does not contain any boxes with number i.
5) For every other row r in the same subblock there is a box with number i.
6) For every other column c in the same subblock there is a box with number i.

Fig. 4 gives an illustration of a valid position for model 2. The position $(9, 7)$ is a valid position for a robot 4. It is clear that on each other row in the same sub-block (row 7 and 8) and oc each other column in the same sub-block (column 8 and column 9) there is a cell containing number 4.

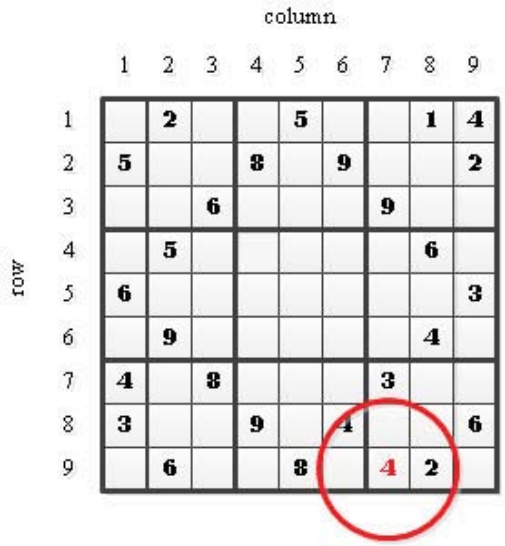

Fig. 4. An example of valid position for model 2.

C. Model 3

For model 3, we use the following definition for a valid position.

Definition 3 A robot i will call a cell at (x, y) a valid position if all the following conditions hold:

1) It is empty.
2) The row x does not contain any boxes with number i.
3) The column y does not contain any boxes with number i.
4) The subblock where (x, y) is located does not contain any boxes with number i.
5) For every other row r in the same subblock there is a box with number i or the cell at the position (r, y) is not empty.
6) For every other column c in the same subblock there is a box with number i or the cell at the position (x, c) is not empty.

Fig. 5 gives an illustration of a valid position for model 3. Similar to our previous example for model 2, the position $(9, 7)$ is a valid position for a robot 4. Although there is no cell containing number 4 in the row 7, the robot 4 still can find this valid position since the position $(7, 7)$ is not empty.

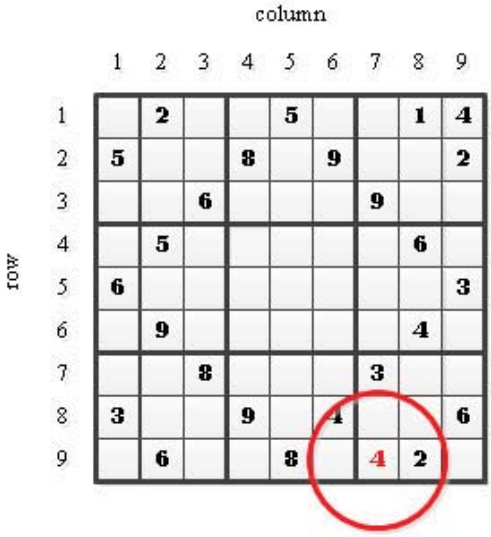

Fig. 5. An example of valid position for model 3.

D. Model 4

In this work we introduce a new model. This model is an improvement of model 3. We put more constraints in the definition of a valid position for a robot. The definition of a valid position of model 4 is as follows.

Definition 4 A robot i will call a cell at (x, y) a valid position if all the following conditions hold:

1) It is empty.
2) The row x does not contain any boxes with number i.
3) The column y does not contain any boxes with number i.
4) The subblock where (x, y) is located does not contain any boxes with number i.
5) For every other row r in the same subblock there is a box with number i or the cell at the position (r, y) is not empty or all cells in row r in the same sub-block with the cell (x, y) are not empty.
6) For every other column c in the same subblock there is a box with number i or the cell at the position (x, c) is not empty or all cells in column c in the same sub-block with the cell (x, y) are not empty.

Fig. 6 shows an illustration of a valid position for model 4. The position $(7, 4)$ is a valid position for a robot 5. Although

there is no cell containing number 5 in the row 8 and column 6, the robot 5 still can find this valid position since all cells in the same sub-block with $(7, 4)$ which lie in row 8 and column 6 are not empty.

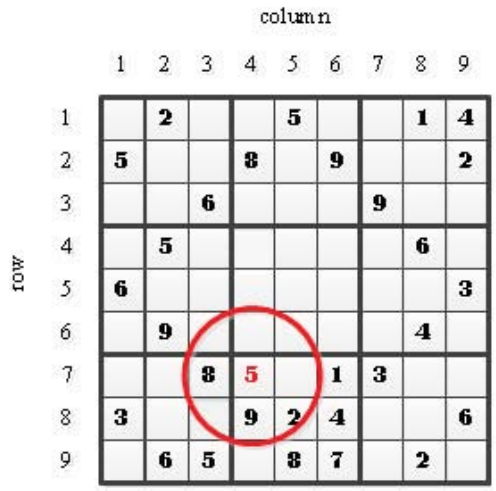

Fig. 6. An example of valid position for model 4.

III. Implementation

In [11] we have modeled Sudoku solver as a parameterized multi agent system. A parameterized multi agent system is a system consisting of a number of similar components (subsystems) that work together, the number of the components is expressed as input parameter of the system. The agents are the robots, the environment are the board and the boxes. The models are written in formal style using TLA+ as specification language.

There are some important aspects that must be considered when implementing multi-agent systems, such as system architecture, inter-agent communication, and agent-scheduling. In this work, we have not yet considered all those issues. We took a simple approach in developing this solver, since the focus of this work is the performance of the solver and not the multi-agent system issues. We used object-oriented programming approach and used Java programming language for developing the solver.

The class diagram of the solver is given in Fig. 7. The solver consists of nine classes, namely Sudoku, Scheduler, Robot, Environment, Position, Box, Cell, List, and ListElement. The explanation of each class is as follows:

- Sudoku class is the main class.
- Scheduler is a class that responsible for controlling the robots or deciding when a robot has its turn.
- Box represents a box. It contains information about the number of the box.
- Cells will store information about a position on the board and the box that on that position.
- Environment stores information about Sudoku board, piles of boxes outside the Sudoku board, and list of the empty cells. Sudoku board is organized as a two-dimensional matrix. The type of each board element is Cell.
- The position is a class that contains information about a position on the board.
- List is used to represent a list of empty cells whose elements are ElementList.
- ElementList is a class that stores information about the Box and a position of the board that has been once occupied by the box.
- Robot class contains two important methods, i.e. are *putOn* and *takeBack*. The two methods represent the robots' main tasks: take a box from a pile outside the board and put it to the board, or take a box from the board and return it back to a pile outside the board.

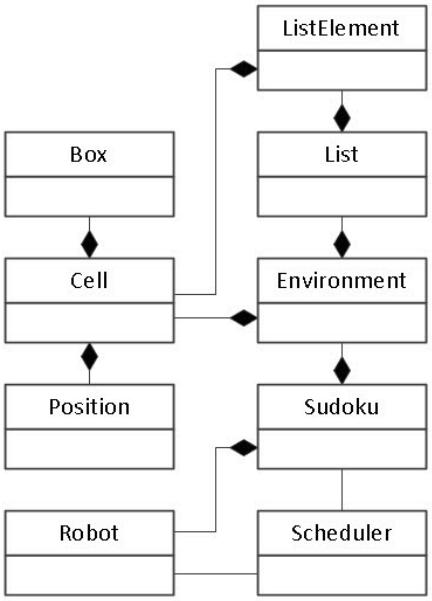

Fig. 7. Class diagram of Sudoku solver.

IV. Experimental results

We tested every model on a hundred puzzles taken from [3] and [12] . Each of the puzzle is guaranteed can be solved or has at least one solution. The number of puzzles that can be solved by each model are given in Table I.

TABLE I
Experimental results 1

Model	The number of puzzles solved
1	100
2	0
3	0
4	44

As expected, model 1 succeeded in finding the solutions of all puzzles. Whereas model 2 and model 3 did not successfully complete any puzzles. Model 4 solved 44 from 100 puzzles. Without considering the failure, in average model 4 needs 4.82 ms for solving a puzzle. The time needed by model 1 for

solving 44 puzzles that can be solved by model 4 is 213 ms or 4.84 ms in average. We may say that whenever model 4 successfully completes a puzzle, the time required for solving the puzzle tends to be smaller than the time required by model 1 for completing the same puzzle.

Furthermore, we measured the performance by considering the success of each model in filling the empty cells. Table II shows the percentage of empty cells successfully completed by each model. Model 1 has the best performance since it managed to fill all the empty cells, i.e. 100%, followed by model 4 and model 3 with 48 % and 22 %, respectively. The success rate of model 2 in filling the empty cells is very small, which is only 1.4 %.

TABLE II
EXPERIMENTAL RESULTS 2

Model	Percentage of filled cells
1	100
2	1.4
3	22
4	48

So far the model 1 has the best performance, followed by model 4. Previous experiment revealed that model 4 is likely to complete the a puzzle faster than model 1. Unfortunately, model 4 is not always successful in completing a puzzle. For the last experiment, we combined model 4 and model 1 to know whether there is a performance improvement. We took 10 hardest puzzles that can only be solved by model 1 based on the time required to complete the puzzles. The combination is done by first letting the model 4 work until it stops, and then running the model 1 to complete the puzzle. Table III shows the time comparison between model 1 alone and the combination of model 4 and model 1 in ms. It can be seen that the combination works better than model 1 alone.

TABLE III
EXPERIMENTAL RESULTS 3

Puzzle	Model 1	Model 4 and 1
1	390	390
2	312	312
3	250	250
4	234	202
5	203	203
6	156	156
7	140	16
8	125	31
9	94	46
10	47	47
Total	1951	1653

V. CONCLUSION

In this paper, we have presented the development of a Sudoku solver. This solver is an implementation of four block-world problem based Sudoku solver models. The experimental results show that the model 1, which uses backtracking principle, has the best performance. Moreover, performance improvement can be achieved by combining model 4 and model 1.

The Sudoku solver models used in this work are taken from our previous work. In [11] the solver is modeled as a block-world problem. In particular, it is represented as a parameterized multi-agent system. In this work, we have not considered the issues related to multi-agent systems. We plan to develop a Sudoku solver simulator that applies the principles of multi-agent systems. This simulator is not only for solving Sudoku puzzles, but also for learning about multi-agent systems, especially about agent communication and scheduling.

REFERENCES

[1] A. Bartlett, et.al. *An Integer Programming Model for the Sudoku Problem*, The Journal of Online Mathematics and Its Applications: Volume 8, Issue May, 2008.
[2] X.Q. Deng & Y.D. Li *A novel hybrid genetic algorithm for solving Sudoku puzzle.*, Optimization Letters, February 2013, Volume 7, Issue 2, pp 241-257, Springer.
[3] Wayne Gould. *Su Doku Junior*. Penerbit Erlangga. 2005.
[4] R. Lewis, *Metaheuristics can solve Sudoku puzzles*, Journal of Heuristics, Vol. 13, 2007, pp. 387-401.
[5] I. Lynce & J. Ouaknine. *Sudoku as a SAT problem*, Proc. of the 9th Symposium on Artificial Intelligence and Mathematics, 2006.
[6] T. Mantere. *Solving, rating and generating Sudoku puzzles with GA.*, Proc. of IEEE Congress on of Evolutionary Computation, 2007. CEC 2007, Singapore.
[7] P. Malakonakis, et.al. *An FPGA-based Sudoku Solver based on Simulated Annealing methods.*, Proc. of International Conference on Field-Programmable Technology, 2009. FPT 2009.
[8] V. Mladenov, et.al., *Solving Sudoku Puzzles by using Hopfield Neural Networks*, Proc. of ICACM'11 Proceedings of the 2011 international conference on Applied & computational mathematics, pp.174-179 World Scientific and Engineering Academy and Society (WSEAS) Stevens Point, Wisconsin, USA, 2011.
[9] J. Monk, et.al. *Solving Sudoku using Particle Swarm Optimization on CUDA*, Proc. of The 2012 International Conference on Parallel and Distributed Processing Techniques and Applications, 2012,.
[10] A. Moraglio & J. Togelius, *Geometric Particle Swarm Optimization for the Sudoku Puzzle*, Proc. of Conference in Genetic and Evolutionary Computation, GECCO 2007, Proceedings, London, England, UK, July 7-11, 2007. ACM 2007.
[11] C.E. Nugraheni & L. Abednego. *Modelling Sudokus as Block World Problems*. International Journal of Computer, Information, Systems and Control Engineering, volume 7, no. 8, pp. 48-54, World Academy of Science, Engineering and Technology, 2013.
[12] Thomas Ag. S. *Sudoku - Edisi Lengkap dengan 270 Aneka Teka-teki Sudoku*. Penerbit Andi. 2012.
[13] T.-W. Yue & Z.-C. Lee, *Sudoku Solver by Qfron Neural Networks*, Proc. of International Conference in Intelligent Computing 2006, ICIC2006, LNCS 4113, pp.943-952, 2006, Springer-Verlag, Berlin Heidelberg 2006.