Exploring A Better Way to Constraint Propagation Using Naked Pair

Kaiqi Chen

Faculty of Computer Sciences, Dalhousie University, Halifax, Nova Scotia, B3H 1W5, Canada

Abstract. Sudoku is an NP-complete problem therefore developing various efficient algorithms is crucial. This paper presents an enhanced approach to solving sudoku puzzles by improving on recursive backtracking with constraint propagation and bitmask, focusing on the implementation of the naked pair technique. This research aims to add constraints to reduce the backtracking steps in solving sudoku using naked pairs, which is a technique that eliminates candidates of a cell from other cells in the same row, column or sub-grid when two cells share the same candidate sets. This research uses a dataset of 1 million sudoku puzzlers from Kaggle to evaluate the proposed solver's performance. The solver first processes the puzzle into bitmask vectors then uses the singles and naked pairs technique to minimize the candidates, and then uses depth-first search to backtrack and solve the puzzle. As a result, the naked pair strategy slightly increases the total steps, and it significantly reduces the computation of depth-first search steps, making the solver potentially more efficient in solving difficult puzzles.

1 Introduction

Sudoku is a fun logical puzzle game that has been played worldwide in the past century. Beyond its nature as a brain training game, sudoku is also a challenging computational problem that has been proven to be NP-complete [1, 2]. The complexity of solving Sudoku highlights the importance of developing efficient algorithms for practical problems, especially in fields like image encryption, optimization and artificial intelligence, where computational tasks often involve similar complexity [3, 4]. Studying in advancing on sudoku solving problems could make sudoku logic applied in those fields more efficient. As the difficulty of solving sudoku puzzles grows in dimensions, the need for enhancing current solving strategies arises as well [5].

The study of how to solve sudoku has lasted for decades and developed across different domains. The traditional ways of human approaches use techniques based on pencilling, which requires the player to write down the possible numbers in each empty cell based on its corresponding row, column and block [6]. This approach narrows down the possibilities greatly but updating the pencil marks is relatively heavy writing for humans to do as it needs to update the table after every step. From the computational side of solving sudoku, the algorithms for solving Sudoku mainly include backtracking and constraint propagation.

Corresponding author: kq712672@dal.ca

© The Authors, published by EDP Sciences. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

Backtracking is a brute-force way that tries to fill the entire table and keep using different possible numbers until meets a valid result. Constraint propagation on the other hand tries to apply different rules to reduce the possibilities of each cell and then propagates on the left possibilities until a solution is found. It is natural to combine the pencilling technique with constraint propagation to reduce the possible backtracking, which has been thoroughly evaluated by many researchers and proved helpful in eliminating possibilities. Keith George Ciantar and Owen Casha have dived deeper into this combination and improved on it using bitmask to reduce memory complexity and computation difficulty [7].

This paper aims to improve Recursive Backtracking with Constraint Propagation using the bitmask algorithm by introducing the naked pair technique. The goal of this research is to reduce the computational steps required to solve Sudoku puzzles, making the process more efficient. To achieve this, this work will first review and present current Recursive Backtracking with Constraint Propagation using the bitmask algorithm. Additionally, the work will demonstrate and implement the naked pair technique under the constraint of third order sudoku puzzle (9*9 table), while taking advantage of the bit vector. After implementing such a design, the performance and comparison with other algorithms such as standard backtracking and constraint propagation will be analysed as well.

2 Approach

2.1 Datasets

Prior to implementing the solver, a reliable dataset is needed to test and evaluate the results. This research uses the public notebook 1 million Sudoku games dataset from Kaggle, which includes 1 million sudoku puzzles and corresponding solutions (https://www.kaggle.com/datasets/bryanpark/sudoku/data). This dataset offers a variety of puzzles with different levels of difficulty which is suitable for testing this paper's solver.

All the puzzles and solutions have 81 digits representing the 9x9 Sudoku grid, listed row by row. Each digit in the range 0 to 9 corresponds to a cell in the grid. A 0 digit indicates that the cell is empty and digits from 1 to 9 represent pre-filled cells (given clues). For example, input "0043 0020 9005 0090 0107 0060 0430 0600 2087 1900 0740 0050 0830 0060 0000 1050 035 0869 0042 9103 00" converts to the grid (shown in Fig1.a) and its corresponding solution "8643 7125 9325 8497 6197 1265 8434 3619 2587 1986 5743 2257 4839 1668 9734 1257 1352 8694 5429 16378" is translated to Fig1.(b) in regular sudoku puzzle's format. In order to process these long arrays and prepare them for testing, the puzzles need to be processed into normal 9x9 two-dimensional integer arrays. This array structure provides a better visualization and allows for more straightforward logical evaluations.

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

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

Fig. 1. Example input (left) and output (right) (Photo/Picture credit: Original)

2.2 Methodology

The approach of this paper builds on the method proposed by Keith and Owen [7], which uses nine bitmask vectors for a row, nine for a column and nine for a 3x3 sub-grid. Each mask vector is 9-bit long, where 1 indicates the presence of the number in the corresponding row, column and sub-grid, and 0 indicates the absence. As shown in Fig. 2, the example grid is transferred to its corresponding mask vectors, which are then used to track potential candidates for each cell. After iterating over the converted 9x9 array, a total of twenty-seven mask vectors are evaluated for solving the puzzle. These mask vectors will be used for the following sudoku-solving strategies. In general, the solver that this work uses first uses singles and naked pairs to reduce the possibilities of valid candidates, then uses a depth-first search to complete the rest of the puzzle.

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

1	0	0	0	0	1	1	1	0

(a) Row Mask

0	0	0	1	0	0	0	0	1

(b) Column Mask

0	0	1	0	1	1	0	0	0

(c) Sub-grid Mask

Fig. 2. Example masks for the first row (a), column (b) and sub-grid (c).(Photo/Picture credit : Original)

2.2.1 Singles

A cell has a “single” if there is only one number that is a valid candidate within its corresponding units (row, column and sub-grid) [8-10]. Fig.3 demonstrates a single in the grid, it can be observed that the highlighted row and column contain 1, which means that crossed sub-grid 1 is only possible to be placed in one cell. This solver uses bitwise AND between the row, column and sub-grid mask vectors and stores the result in the result vector. The resulting vector is the candidate of that cell. If there is only one bit left in the result vector, which means there is only one possible candidate, that bit represents the hidden single. Once a single is found, the solver finds the position of that bit, and then updates the result grid by adding the position it found in the corresponding cell. It then updates this digit in the unit mask vectors using bitwise AND with a mask to clear the appropriate bit.

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

Fig. 3. Example of a single in the puzzle(Photo/Picture credit : Original)

2.2.2 Naked Pair

Naked pair is a slightly more advanced sudoku-solving skill to eliminate the possible candidates. The benefit of the naked pairs strategy is when two cells of the same units have the same two possible candidates, these two candidates can be eliminated from every other cell in the corresponding row, column or sub-grid [8-10]. An example is shown in Fig.4, note that cells (9, 5) and (9, 6) both have possible candidates 2 and 6, which means all other cells in that column cannot contain 2 or 6, thus eliminating the possibility of 2 in cell (9, 8) leaving 4 as the final candidate of that cell. The process of finding naked pairs in this algorithm involves examining the candidate sets of empty cells in each unit, which uses a series of bitwise operations to detect pairs that have the same two candidates.

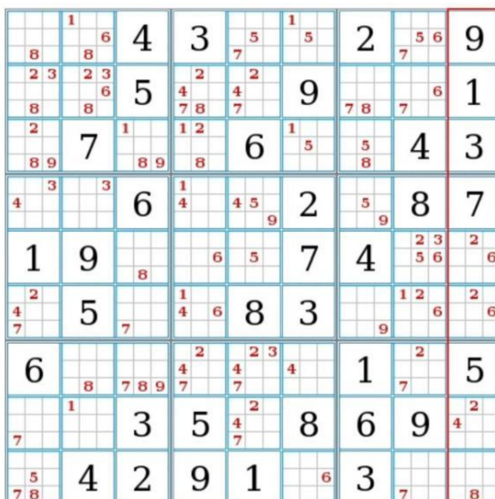

Fig. 4. Example of naked pairs(Photo/Picture credit : Original)

First, the solver iterates over the empty cells in the grid. For each cell, it evaluates the cell's candidate set, which is represented by a 9-bit integer where each bit represents the possible candidates. The solver then checks the population of candidates in the cell, if the cell has exactly two candidates, it is marked as the potential naked pairs for later comparison. When a potential naked pair is found, it is compared with the previous cells in the same unit that also have two candidates. If an exact match is found, this indicates a naked pair is identified. Then the solver removes these candidates in other cells within the same units and updates the result grid.

2.2.3 Depth-first Search

Although finding singles and naked pairs greatly reduces the possible candidates in the puzzle, it may still not eliminate all possibilities in some harder puzzles. In such a case, backtracking is still needed to complete the puzzle. Depth-first search is a backtracking strategy that is used when above the puzzle can not be further reduced by singles and naked pairs. While depth-first search and breadth-first search are commonly used for backtracking, research has shown that depth-first search tends to be faster for sudoku puzzles, which is why it is chosen for this implementation [11]. This search algorithm traverses the entire grid and tries the remaining candidates in empty cells until it either meets a dead end or completes the puzzle.

To start depth-first searching, an empty cell must be located. This solver scans the grid from top-left to bottom right to find the unfilled cell. Once the empty cell is encountered, the solver makes a guess selects one of the candidates from the candidates set and updates the related units. After placing the guess, the solver recursively applies the same process to the next empty cell. This creates a chain of guesses, each building upon the previous one. A stack is used to track the guess. Each time a guess is made, the cell's position and placed bit is pushed onto the stack. If an empty cell has a candidate set of zero, an invalid guess occurs. In this case, the solver needs to undo the last guess by popping the top entry from the stack to undo the last move.

The above three sudoku-solving strategies are recursively applied as well. While the puzzle is not completed, the main problem-solving process tries to find the singles first. If no singles are found, it then tries to find naked pairs. After trying the two elimination strategies,

a depth-first search is then applied. The same steps are applied until either the puzzle is solved, or the depth-first search takes more than 1000 steps which indicates a non-solvable puzzle.

3 Result

The purpose of this paper is to study the improvements brought by applying naked pairs to the existing sudoku solver algorithm. The factor that this work focused on evaluating the performance is the reduction of depth-first search (DFS) steps required to solve the puzzle. Compared to finding singles or naked pairs, DFS is significantly more time-consuming due to the exhaustive nature of the search process. Since the execution time and memory usage may vary from machine to machine, the number of DFS steps needed to solve a puzzle will be constant if the same solver is applied to the same puzzles. This ensures that the comparisons are consistent regardless of the hardware. Therefore, the comparison focused on the number of DFS steps is critical to studying the efficiency of the solver and a core performance identifier.

To compare the two algorithms, both solvers are tested with one million non-repeated puzzles with and without the naked pair detection enabled. The performance evaluation will focus on the DFS steps needed as well as the consideration of the total number of steps and run time.

Table 1. Comparison step performance with and without naked pair

	Total steps	Average steps per puzzle	Total DFS step
This solver	6416269	6.42	2
Solver without naked pair	6089578	6.09	397

As shown in Table 1, while this paper's strategy takes 6416269 steps in total, compared to 6089578 steps for the solver without naked pairs, the difference is only 5.36%. This is due to the introduction of the naked pair technique that increased the steps for computation. However, the naked pair technique only takes linear time and there is room to optimize this process. In contrast, depth-first search backtracks all possibilities which require significantly more computation, especially in harder puzzles.

This can be observed from the total DFS steps performance. Although the total steps are higher after enabling naked pairs, the total DFS steps have drastically reduced. The solver using the naked pair technique required only 2 DFS steps across all one million puzzles, compared to 397 DFS steps without it. This indicates that the naked pairs technique significantly eliminates the possibility of candidates, making backtracking much more efficient.

Overall, the results highlight the advantages of the naked pair technique while incurs a small increase in total steps and time, it still greatly optimises backtracking, making the solver more efficient in handling more complex sudoku puzzles. Hard or professional sudoku puzzles provide fewer clues in the grid, which makes it more difficult to find singles and results in more guessing which increases the DFS steps, making DFS the more time-consuming steps to solve the puzzle. However, with the introduction of naked pairs, the DFS steps were hugely reduced which resulted in a more efficient solver for difficult puzzles.

4. Conclusion

This research examines the effectiveness of using the naked pair technique in existing sudoku-solving algorithms. The solver first tries to minimize the possible candidates using

the singles and naked pairs strategy, then uses a depth-first search to backtrack and complete the rest of the puzzle. Although using the naked pair leads to a modest increase of 5.36% in total steps, it still significantly reduces the number of depth-first search steps from 397 to only 2. This result highlights the efficiency improvement in solving sudoku puzzles by narrowing down potential candidates and reducing the need for exhaustive backtracking. The proposed solver proved its advantage in solving a sudoku puzzle, it still can be further improved by optimizing the naked pair process and trying more complicated solving techniques.

One way to optimize naked pair detection is to set up an early stopping mechanism. One of the reasons the proposed solver takes more steps in total is that it searches the entire board every time to find naked pairs. Using queues or other data structures to store the searched units may avoid unnecessary whole-board-search and stop the detections early. Another way to improve this is to use parallel processing to find naked pairs across rows, columns and sub-grids by multi-threading or GPU-based parallelism. Besides these possible optimization methods, there is always more room to incorporate more advanced techniques like hidden pairs and naked triples. However, the use of these may result in similar conditions where they may reduce possibilities, but increase the steps taken in the end.

Reference

1. T. Yato and T. Seta, “Complexity and completeness of finding another solution and its application to puzzles,” IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol. E86-A (5), pp. 1052–1060, 2003.
2. E. Hoexum. Revisiting the proof of the complexity of the sudoku puzzle. Diss. 2020.
3. S. A. Mehdi and A. A. Kadhim, "Image Encryption Algorithm Based on a New Five Dimensional Hyperchaotic System and Sudoku Matrix," 2019 International Engineering Conference (IEC), Erbil, Iraq, 2019, pp. 188-193.
4. Y. Björnsson, S. Helgason and A. Pálsson, "Searching for Explainable Solutions in Sudoku," 2021 IEEE Conference on Games (CoG), Copenhagen, Denmark, 2021, pp. 01-08.
5. M. Najafian, T. A. Gulliver and M. Esmaeili, "Construction of Standard Solid Sudoku Cubes and 3D Sudoku Puzzles," in IEEE Access, vol. 10, pp. 30180-30188, 2022.
6. [J. F Crook. "A pencil-and-paper algorithm for solving Sudoku puzzles." Notices of the AMS 56.4 (2009): 460-468.
7. K. G. Ciantar and O. Casha, "Implementation of a Sudoku Puzzle Solver on a FPGA," 2023 9th International Conference on Control, Decision and Information Technologies (CoDIT), Rome, Italy, 2023, pp. 1803-1808.
8. P. Berggren and D. Nilsson. "A study of Sudoku solving algorithms." Royal Institute of Technology, Stockholm (2012).
9. D. M. Pais, "Sudoku Solver Based on Human Strategies an Application in VBA." Order No. 30653370, Universidade de Lisboa (Portugal), Portugal, 2022.
10. N. Pilavakis, Sudoku Solver, [online] Available: https://project-archive. inf. ed. ac. uk/ ug4/20212573/ug4proj.pdf.
11. N.M. Diah, et al. "Sudoku solutions: a comparative analysis of breadth-first search, depth-first search, and human approaches." Journal of Education and Learning (EduLearn) 19.1 (2025): 561-569.