World Journal of Advanced Research and Reviews
eISSN: 2581-9615 CODEN (USA): WJARAI
Cross Ref DOI: 10.30574/wjarr
Journal homepage: https://wjarr.com/

(RESEARCH ARTICLE)

Check for updates

Local search for solving sudoku puzzles

Blessing Ngara * and Linsey Kitt

Department of Computer Science, Iowa State University, Ames, IA, USA.
Blessing Ngara; ORCiD: 0009-0006-0033-0829.
Linsey Kitt; ORCiD: 00009-0004-5490-0087.

World Journal of Advanced Research and Reviews, 2026, 30(02), 1294-1298

Publication history: Received on 10 April 2026; revised on 10 May 2026; accepted on 12 May 2026

Article DOI: https://doi.org/10.30574/wjarr.2026.30.2.1307

Abstract

Sudoku puzzles, logic puzzles traditionally represented by 9x9 grids, have been studied by many researchers looking to understand different types of search approaches better, especially as Sudoku puzzles are NP-complete for variable-sized grids. Many papers have focused on constraint- satisfaction to solve Sudoku puzzles with good success, using the constraints that each number one through nine can only appear once in any given column, row, or 3x3 sub-square for a 9x9 Sudoku puzzle. In contrast, this paper looks to explore the effectiveness of using local search algorithms to solve Sudoku puzzles. Here, we present our strategy and experience implementing four local-search algorithms using two different search heuristics and their ability to solve a set of 9x9 Sudoku puzzles of varying difficulty. Additionally, we provide discussions on our results and an overview of the challenges and successes we had.

Keywords: Sudoku; Puzzles; Logic; Algorithm

1. Introduction

Sudoku is a popular brainteaser that challenges players to fill a 9x9 grid with numbers. An example can be seen in Table 1. The grid is further divided into smaller 3x3 grids, as can be seen in the table. The objective is to place the digits 1 through 9 in each row, column, and 3x3 grid exactly once. A puzzle may begin with a set of numbers already assigned to specific cells, which cannot be changed. A well-crafted Sudoku puzzle will only have one unique solution that satisfies all these placement rules and maintains the preset cells. The difficulty of puzzles can be described in a few ways. Common terms like “easy,” “medium,” and “hard” offer a general idea of how challenging they are to solve by hand. Addition- ally, the complexity of published puzzles can be measured by the time it takes for computer algorithms to find a solution or by the number of steps the algorithm takes to solve it.

Solving N x N size Sudoku puzzles is an NP-complete problem (YATO and SETA 2003), so solving them with various search algorithms is a topic of interest to researchers. Many researchers use constraint-solving for Sudoku puzzles (Simonis 2005; Musliu and Winter 2017; Lewis 2007; Reeson et al. 2007). Constraint Satisfaction

* Corresponding author: Blessing Ngara

Table 1 An example of a Sudoku puzzle to be solved by our algorithms

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

Problems (CSPs) are a class of problems in artificial intelligence where the goal is to find an assignment of values to variables that satisfy a set of constraints (Russell and Norvig 2010). These constraints define relationships that must hold true between the variables within a certain domain. For example, in a Sudoku puzzle, variables are the numerical values assigned to each cell, while the constraints are the requirements for each number to appear exactly once in any given row, column, and 3x3 grid. There are various techniques for solving CSPs which include backtracking, constraint propagation, and heuristics. By utilizing different solving techniques and heuristics, CSP algorithms can efficiently find solutions that satisfy all the defined constraints. Instead of looking at constraint-solving, we wanted to take a different approach to solving Sudoku puzzles. In this paper, we instead investigate local search algorithms for solving Sudoku puzzles. Local search algorithms are a well-established family of iterative optimization techniques employed to tackle a wide range of optimization problems. They function by exploring a solution space and progressively converging towards increasingly optimal solutions. Each point within this search space represents a candidate solution. The algorithm starts at an initial solution and then iteratively evaluates neighboring solutions based on a pre-determined heuristic function. If a neighboring solution exhibits a heuristic function value which is closer to the optimum value, the algorithm moves to that new state, replacing the current solution. The iterative process continues until it reaches a predetermined limit, which is called the stopping criterion. In this paper, the stopping criterion varies depending on the algorithm used and the number of states visited per iteration.

This paper will investigate the implementation of various local search algorithms for solving Sudoku puzzles (Rus- sell and Norvig 2010). These algorithms include first-choice hill-climbing, simulated annealing, local-beam, and genetic algorithms. Hill climbing is a basic local search technique that iteratively explores the solution space. At each step, it evaluates neighboring solutions, possible Sudoku configurations, and moves towards the one with the highest objective function value. This process continues until a local optimum is reached, where no better neighbors exist. In our case, be- cause Sudoku puzzles have a large search space, we use first- choice hill-climbing, where instead of looking at all possible neighboring Sudoku configurations and selecting the best, we randomly generate neighboring Sudoku configurations until a better one is found to move to. The simulated annealing algorithm is inspired by the physical process of annealing in metallurgy; this algorithm introduces a probabilistic element to escape local optima. While it still favors better solutions, it allows occasional moves to "worse" neighbors with a controlled probability. This probability gradually decreases as the search progresses, mimicking the cooling process in annealing. This allows the algorithm to explore a wider search space and potentially find the global optimum. Local-beam search maintains a pool of promising solutions during the search. At each iteration, it evaluates all the neighbors of all solutions in the beam and selects a predefined number of the most improved ones to replace the cur- rent beam. This allows for a more diverse exploration of the search space compared to hill-climbing, potentially leading to better solutions. Lastly, we implemented a genetic algorithm, which is inspired by biological evolution. This algorithm operates on a population of candidate solutions rep- resented as chromosomes. It iteratively performs crossover, where two parent chromosomes are combined to form child chromosomes; mutation, where chromosomes are randomly modified; and selection, where the best chromosomes are selected for the next generation. This process aims to evolve the population towards solutions that increasingly satisfy the Sudoku puzzle.

In the remainder of this paper, we first go over our methodology for setting up our experiment. Following this, we go over the results of our experiment and provide discussion on our results. We then conclude with a summary of our work and what future work could be done to further the project.

2. Methodology

To implement various local search algorithms for solving Sudoku puzzles, we used outlines of four basic algorithms and formatted them for Sudoku puzzles (Russell and Norvig 2010). These algorithms include hill climbing, simulated annealing, local-beam, and genetic algorithms. We included two different heuristics to determine the quality of a puzzle solution, the first being a basic incremental heuristic that determines the quality by the number of unique entries one through nine in each row, column, and 3x3 sub-grid. Our second heuristic was similar, except instead of just being the number of unique entries. It was the number of unique entries with a bonus of two added for each row, column, and 3x3 sub-grid that had at least seven unique entries. This second heuristic then gives a bonus when puzzles include rows, columns, and 3x3 sub-grids that are almost optimal.

To represent our Sudoku puzzles, we first read from a file that included a set of empty cells and cells that already had a number assigned. Each empty cell was given a random number one through nine that was not already included in that row. Each cell that was already defined in the file was marked as a preset, so it could not be changed by the algorithms. By instantiating our puzzles in this way, we could operate on them by swapping numbers within each row, al- lowing each row to contain each unique number, so only columns and sub-grids need to be solved.

We wrote each algorithm in Java and created a script to run each of our algorithms on a set of 11 Sudoku puzzles read from files 20 times with each heuristic, for a total of 40 times per puzzle per algorithm. We organized our results, which counted the number of puzzles explored to reach a solution and the number of puzzles solved, in a table and used this to determine which algorithm performed the best.

3. Experiments

The results of our experiment can be found in Table 2. The data is split into each category of algorithm, heuristic, and puzzle in the given run. In total, each algorithm attempted to solve Sudoku puzzles 440 times, 20 times per each of the two heuristics per each of the 11 puzzles. We tracked how many times each algorithm was able to solve a puzzle and the average number of states explored to find the solution, which is shown in the table. Sometimes, an algorithm was never able to find a solution to a given puzzle, in which case the number solved is zero and the average number of states is NA. Analysis of our results is in the following section.

4. Discussion

First, as seen in Table 2, simulated annealing had the best performance out of the four algorithms as it solved 352 out of the 440 runs. This is a success rate of 80%, which we believe is a result of the fact that simulated annealing incorporates a random element that allows it to escape local optima. This means it can jump out of situations where a seemingly good solution candidate might not be the best overall. This exploration allows it to find better solutions across various puzzle runs. The algorithm that was second in number of puzzles solved was local beam, which solved 14 out of the 440 runs. It was still able to see some success, but this is clearly a much lower rate than simulated annealing. The low success rate of 3.2% may be because local beam is not as efficient at escaping out of local optima in this type of problem. We did incorporate some ability for beams with worse fitness to be used, but this was not enough to escape the local optima in most cases. Next, hill climbing had a similarly low performance: it only solved six out of the 440 runs. Our hill climbing implementation offered no random move to puzzle guesses with worse solutions, so hill climbing's success rate of 1.4% was likely the result of it only considering solutions that look better than the current. It had no opportunity to escape the local optima. Finally, the genetic algorithm did not solve any puzzles, demonstrating the worst performance out of the four algorithms. The algorithm's success rate of 0% might be because genetic algorithms are better suited for problems with a clear definition of "good" and "bad" solutions and the ability to combine elements from different solutions. Puzzle solving may lack this clear definition, and the crossover operation might not always lead to better results. Additionally, the specific configuration of the genetic algorithm for these puzzles might not have been ideal: some of the random parameters such as the setup of crossover may have more optimal setups for the context of solving Sudoku puzzles.

Table 2 Number of times a Sudoku puzzle was solved out of 20 attempts and the average number of states explored to solve the puzzle with four algorithms used with two heuristics

	Hill Climbing				Simulated Annealing			
	Heuristic 1		Heuristic 2		Heuristic 1		Heuristic 2	
	Num. Solved	Avg. Num. of States	Num. Solved	Avg. Num. of States	Num. Solved	Avg. Num. of States	Num. Solved	Avg. Num. of States
Puzzle 1	0	NA	0	NA	9	5,689,499	6	5,098,540
Puzzle 2	0	NA	0	NA	19	4,593,040	19	4,556,828
Puzzle 3	0	NA	0	NA	20	3,718,591	20	3,590,762
Puzzle 4	0	NA	0	NA	20	4,847,106	20	4,626,635
Puzzle 5	0	NA	0	NA	20	3,778,548	20	3,901,611
Puzzle 6	3	3,221	2	6,618	20	3,326,674	20	3,361,695
Puzzle 7	0	NA	0	NA	7	5,702,024	8	6,127,307
Puzzle 8	0	NA	0	NA	20	4,614,403	19	4,790,480
Puzzle 9	1	1,350	0	NA	20	3,559,789	20	3,641,674
Puzzle 10	0	NA	0	NA	18	5,502,704	17	5,566,980
Puzzle 11	0	NA	0	NA	5	6,789,595	5	6,719,120
	Local-Beam				Genetic			
	Heuristic 1		Heuristic 2		Heuristic 1		Heuristic 2	
	Num. Solved	Avg. Num. of States	Num. Solved	Avg. Num. of States	Num. Solved	Avg. Num. of States	Num. Solved	Avg. Num. of States
Puzzle 1	0	NA	0	NA	0	NA	0	NA
Puzzle 2	0	NA	0	NA	0	NA	0	NA
Puzzle 3	0	NA	0	NA	0	NA	0	NA
Puzzle 4	1	1,127,130	1	1,220,570	0	NA	0	NA
Puzzle 5	1	2,108,760	1	336,260	0	NA	0	NA
Puzzle 6	4	476,490	2	7,430,415	0	NA	0	NA
Puzzle 7	0	NA	0	NA	0	NA	0	NA
Puzzle 8	0	NA	0	NA	0	NA	0	NA
Puzzle 9	3	4,525,250	1	506,890	0	NA	0	NA
Puzzle 10	0	NA	0	NA	0	NA	0	NA
Puzzle 11	0	NA	0	NA	0	NA	0	NA

Aside from success rates, it can also be seen that some puzzles were clearly easier for algorithms to solve than others. There were several puzzles simulated annealing was able to solve every or nearly every time, while others were only solved a quarter of the time. A few of these puzzles with the highest success rates for simulated annealing also had the highest success rates for hill climbing and local-beam, especially Puzzle 6, which was solved five times out of the six successful runs for hill climbing and six times out of the 14 from local-beam. Puzzles that were solved less often with simulated annealing were not solved by hill climbing or local-beam, and they took more states explored on average with simulated annealing than the more frequently solved, "easier" puzzles.

Looking at the number of states explored to solve a puzzle, the fastest solves were all with hill climbing. This may suggest that beyond these low numbers of states explored, hill climbing remains stuck in a local optimum. If it does not get stuck, it is able to quickly get to the solution since it is not being slowed down by random moves away from the optima. Local beam was also able to solve puzzles faster in some instances than simulated annealing, though not as dramatically so as is the case with hill climbing. This may suggest that though random moves offer a better rate of solving success by escaping local optima, they can slow down the speed at which the solution is found by sometimes moving away from an optimal solution.

Finally, when looking at the performance with different heuristics, we noticed there was not much of a difference across the runs. This leads us to believe the bonus we added to the heuristic had low influence on the way the algorithms navigated the search space.

Future Work

With the results we observed in the previous section, we identified several future paths of exploration to better understand each algorithm and its potential for solving Sudoku puzzles. First, given that the genetic algorithm was unsuccessful for us, we believe there are many different parameter configurations that could be employed to improve its success. We did not have the time to explore them in this work, but we believe it would be worth continued exploration to understand how it can be made to work with Sudoku puzzles. Along this same line of thought, each algorithm has different configurations that could be tried, for example different temperatures or iterations for simulated annealing, a different setup for making ”bad” moves for the local-beam search, or adding random restarts to hill climbing to try to give it more opportunities to start outside of a local optima. These different configurations may be able to solve puzzles more often or solve them faster with fewer states explored.

Another line of future work is to explore more heuristics. The results of the different heuristics in this work were uneventful, but exploring more may give a better idea of how the heuristics affect puzzle-solving and the types of strategies that benefit Sudoku-solving.

5. Conclusion

As seen in this work, we implemented four local search algorithms with two heuristics to attempt to solve 11 Sudoku puzzles. Simulated annealing emerged as the most effective algorithm for solving puzzles of varying difficulty among the four algorithms evaluated. The ability to escape local optima was seen to offer a significant advantage in finding optimal or near-optimal solutions. This is especially apparent comparing simulated annealing to hill climbing, which had significantly worse performance in the number of puzzles solved. Though the local-beam and genetic algorithms also had low performance in this regard, they may still have potential with different setups and parameter configurations than what we used, a potential worth exploring as future work. While each algorithm offers different advantages and disadvantages to solving various search problems, they may have different problems they are best at. Simulated annealing showed clear potential for Sudoku puzzles and was relatively simple to implement, but the others with some more effort may be able to show better success than they have in this paper.

Compliance with ethical standards

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] 2007. An interactive constraint-based approach to sudoku. 1976–1977.

[2] Russell, S., and Norvig, P. 2010. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 4th edition.

[3] Simonis, H. 2005. Sudoku as a Constraint Problem. *CP Workshop on Modeling and Reformulating Constraint Satisfaction Problems* 13–27.

[4] YATO, T., and SETA, T. 2003. Complexity and completeness of finding another solution and its application to puzzles. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* E86-A.

[5] Lewis, R. 2007. On the combination of constraint programming and stochastic search: The sudoku case. In Bartz-Beielstein, T.; Blesa Aguilera, M. J.; Blum, C.; Naujoks, B.; Roli, A.; Rudolph, G.; and Sampels, M., eds., *Hybrid Metaheuristics*, 96–107. Berlin, Heidelberg: Springer Berlin Heidelberg.

[6] Musliu, N., and Winter, F. 2017. A hybrid approach for the sudoku problem: Using constraint programming in iterated local search. *IEEE Intelligent Systems* 32(2):52–62.

[7] Reeson, C.; Huang, K.-C.; Bayer, K.; and Choueiry, B. Y.