<u>Group 36</u>
Isaac Walter
Tyler Foster
Robert Phipps

Artificial Intelligence: Project 1

ABSTRACT

We were assigned to create 5 different artificial intelligence algorithms to solve different level difficulties of sudoku puzzles, those five being simple backtracking, backtracking with forward checking, backtracking with arc consistency, simulated annealing, and genetic algorithm. When looking at our results we were able to get all of the backtracking methods to solve the easy and medium puzzles and we could get the local search methods to solve every difficulty of the puzzles. When it came to local search we were not able to guarantee the algos to be able to solve the evil difficulty problems 100% of the time due to their stochastic nature and out of these 2 methods we found the genetic algorithm to be the least consistent but with careful tuning we got them performing similarly. Out of the backtracking methods we found arc consistency to be the most computationally efficient and was able to solve some harder puzzles unlike the other backtracking methods. We conclude that the optimal algorithm choice is problem-dependent, with AC-3 excelling on less constrained puzzles and simulated annealing representing the most robust solution for extreme difficulties.

PROBLEM STATEMENT

The fundamental challenge of the project was to fill out the sudoku board with the numbers 1-9 and have no repeat numbers in each row, column, and box. We hypothesize that the simple backtracking algo will successfully solve the easy and medium difficulty puzzles. We expect the easier puzzles to have a smaller search space allowing it to be possible for the backtracking method and the harder puzzles will have too large of a search space to be solved. The simple backtracking method can waste computational effort by tuning into dead ends that could have been seen beforehand, this is where backtracking with forward checking comes in. This algo will perform forward checking before assigning a cell with a number to inform the program of any dead ends beforehand. We hypothesize that the backtracking with forward checking will have to make less moves and therefore be less computationally expensive and it will run faster. We think it will also be able to solve the easy and medium puzzle without a problem and possibly some hard puzzles, but not the evil level. If we take forward checking up a notch and create a more robust approach that keeps the whole board consistent at all times we would get arc consistency. We hypothesize that arc consistency will be the best algo with backtracking and will be able to solve all the puzzles due to its dramatic reduction in the amount of explored states necessary to get a solution. Sometimes backtracking is not the best answer to these problems, local search strategies can prove to be better. We hypothesize that simulated annealing will be highly effective at solving all the puzzles given. We believe that making calculated repairs to a board will be more efficient and beneficial to solve a puzzle rather than backtracking; however, its stochastic nature means there is always a possibility it will not find the answer. Another stochastic local search approach is to evolve a solution over generations. We hypothesize that the genetic algorithm will be the most efficient algorithm we make and will be able to solve puzzles of all difficulties. We think that the most important factor will be tuning the hyperparameters so we can get the best results from the algo but this is also stochastic and will not have a 100% success rate.

DESCRIPTION OF ALGORITHMS

The simple backtracking method will look for an empty cell and then place a number 1-9, check if it is valid and continue with that process. If this algo reaches a dead end it will backtrack to get unstuck using recursion.

The forward checking algo is very similar to the ladder; however, it will check the domain of possible values by looking at the neighboring box, column, and row and calculate to see if it leads to a deadend then will place a number within the domain therefore, making a more efficient algorithm.

The next algo is backtracking with arc consistency which starts by choosing an empty cell and blindly inputting a number 1-9. After the previous process it will then read the given numbers for the puzzle and rule that number out of the domain for its box, row and column. We then check if any cells have an empty domain, if yes this means the puzzle cannot be solved and we recurse back to placing a number in the empty cell, if no we recurse to the next and repeat the process till the puzzle is solved.

The simulated annealing algorithm starts by creating an initial candidate Sudoku board that respects the given numbers and fills the empty cells randomly while maintaining valid rows. The system then iteratively selects a cell that is involved in a conflict and attempts to change its value to one that minimizes the number of conflicts, following the minimum conflict heuristic. After each proposed change, the system evaluates whether to accept the new board state. If the new state reduces conflicts, it is always accepted; if it increases conflicts, it may still be accepted based on a probability that depends on the current temperature. The temperature is gradually lowered according to a “cooling” function, reducing the likelihood of accepting worse solutions over time. This process continues, selecting new conflicting cells and evaluating changes, until the puzzle is solved.

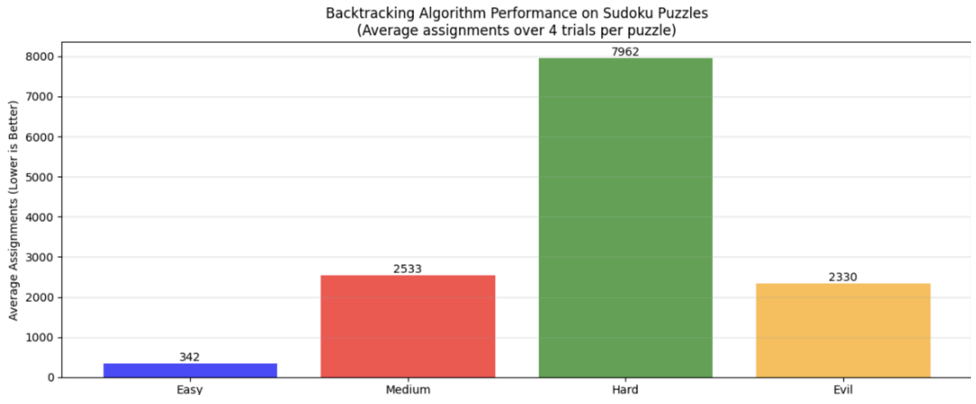
For the genetic algorithm it starts by creating a population of randomly generated sudoku boards based on the one provided that will have each row have the numbers 1-9 (the boxes and column will be fixed later). Each one will be processed to check if they fit the rules of the puzzle to see if one is correct (if one is correct we print out the puzzle). We measure how close the puzzle was to being correct with a penalty function. We select the puzzles that had the best penalty score and make them the “parents”. We do a crossover with 2 randomly selected parents (example: by taking half of the rows of parent A and combining them with half the rows of parent B) then mutate them (making random small changes to the crossover) based on our hyper parameter “mutation_rate”. We take these new crossovers and mutated children and they become our new generation and will run through the same process until we find a puzzle with 0 penalties.

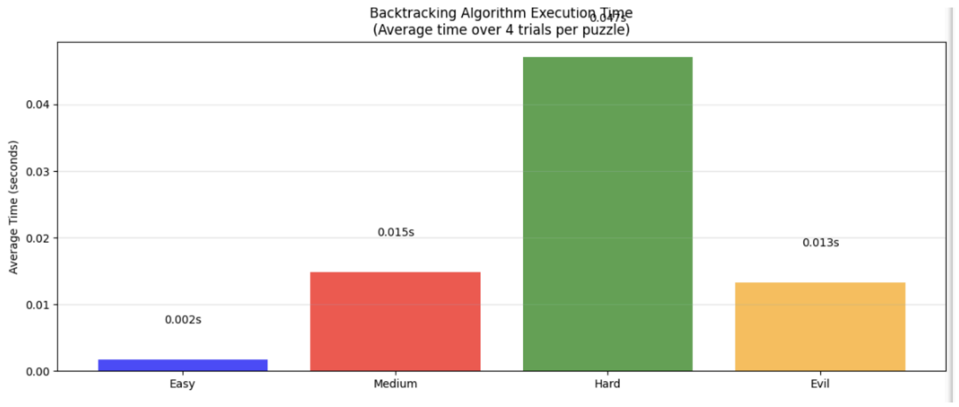
EXPERIMENTAL APPROACH

Our approach to this project was to systematically evaluate the effectiveness of each algo we will be making by testing them across a set of 16 different puzzles varying in difficulty. We will analyze the algo’s carefully to see key performance metrics like solve time, number of backtracks (specifically for the backtracking algos), number of generations (specifically for the local search algos), and success rate. For the local search algos we want to look closely on how adjusting the hyperparameters affects the program's results to therefore give us the best results. By using the same input puzzles against all 5 of the algos we aim to create data that will objectively show the strengths and weaknesses of all the programs and figure out which ones are the best for solving which difficulty of the problem.

OUR RESULTS

Backtracking

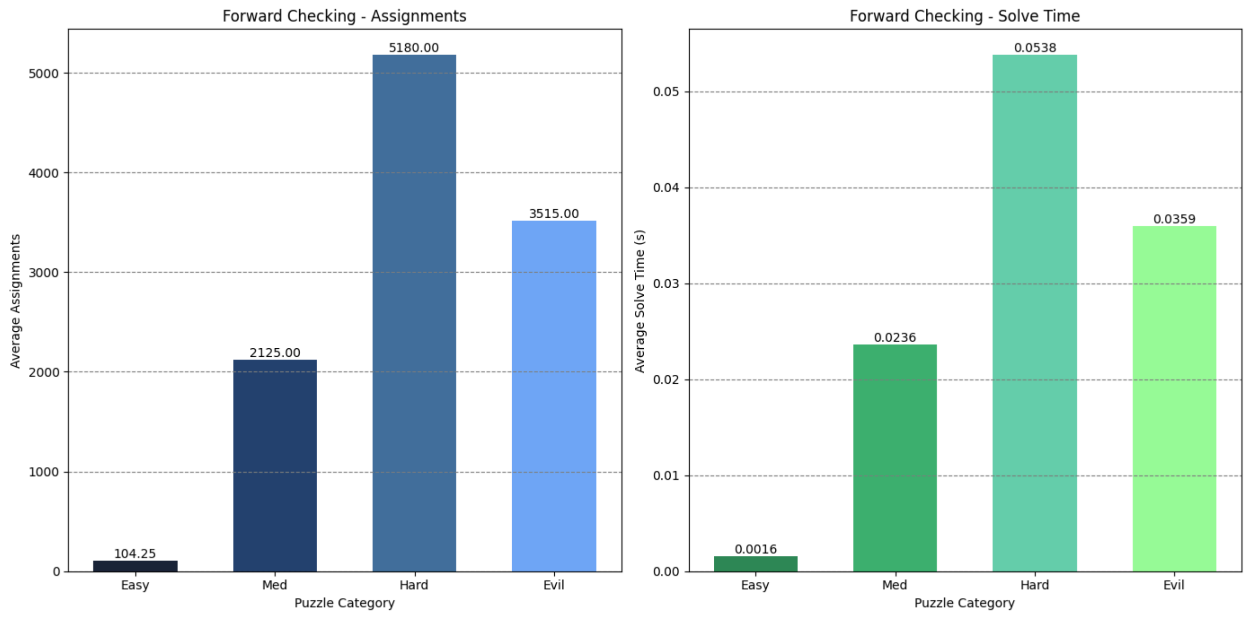




The graphs above represent the simple backtracking algorithm we made. The top one shows the average assignments made between each difficulty of the puzzle and the bottom one shows the average time it takes per assignment. The information displayed is very much how we thought it would turn out besides the stats for the evil difficulty puzzles. The Evil puzzles were ⅓ the size in total assignments and assignment time but this could be due to the complexity of the puzzles being so advanced that the simple backtrack would face contradiction so fast that it would terminate that branch of development very quickly thus causing the results we see in the plots. Simple backtracking performed the best for solving the easy puzzles by far with a .002 second assignment time and 342 average total assignments which is very impressive.

Forward Checking

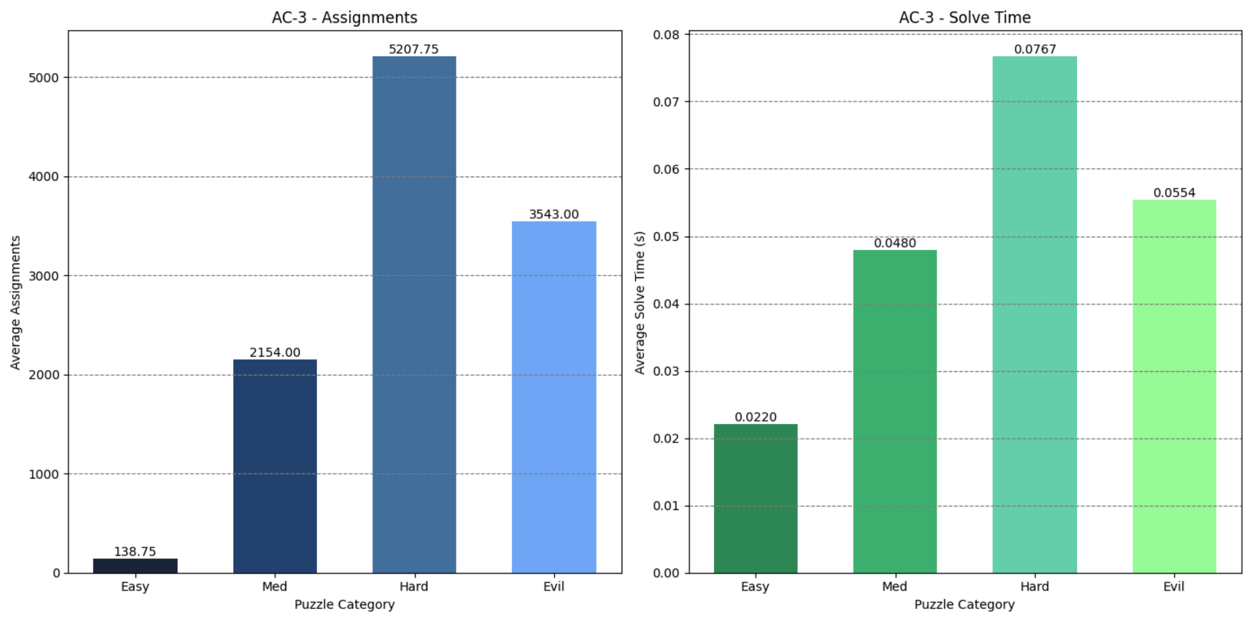
Picture 2 shows the results for the forward checking algorithm. The left side displays the average assignments for each puzzle difficulty, while the right side shows the average solve time. Forward checking generally made fewer assignments than just backtracking, especially for medium and hard puzzles, which suggests that the algorithm's early detection of conflicts helped out a bit. For easy puzzles, the assignment count and solve time were similar to backtracking, but for the evil and hard puzzles, both values decreased a lot. This is likely because forward checking identified conflicts in the hard and evil puzzles, causing the algorithm to adjust early.



Picture 2: Average values over 4 tests for each puzzle

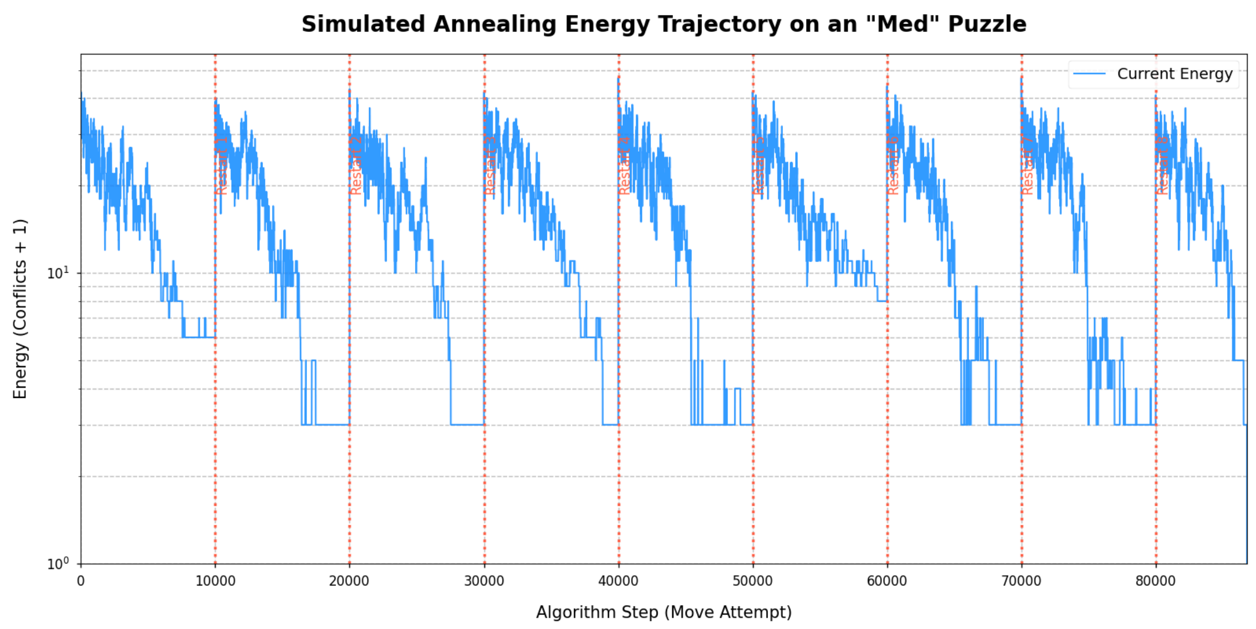
AC-3

Picture 3 shows the AC-3 algorithms performance. Once again, the left graph shows the average assignments per category and the right graph shows the average solve time. AC-3 did well on easy and medium puzzles, thanks to its constraint propagation. For hard and evil puzzles, the average assignment was on par with simple backtracking and forward checking, but its solve time was significantly higher. This is probably due to the extra time spent propagating constraints. This method seems effective for simpler puzzles, but less so for complicated ones. The more obvious the constraint, the better for AC-3.



Picture 3: Average values over 4 tests for each puzzle

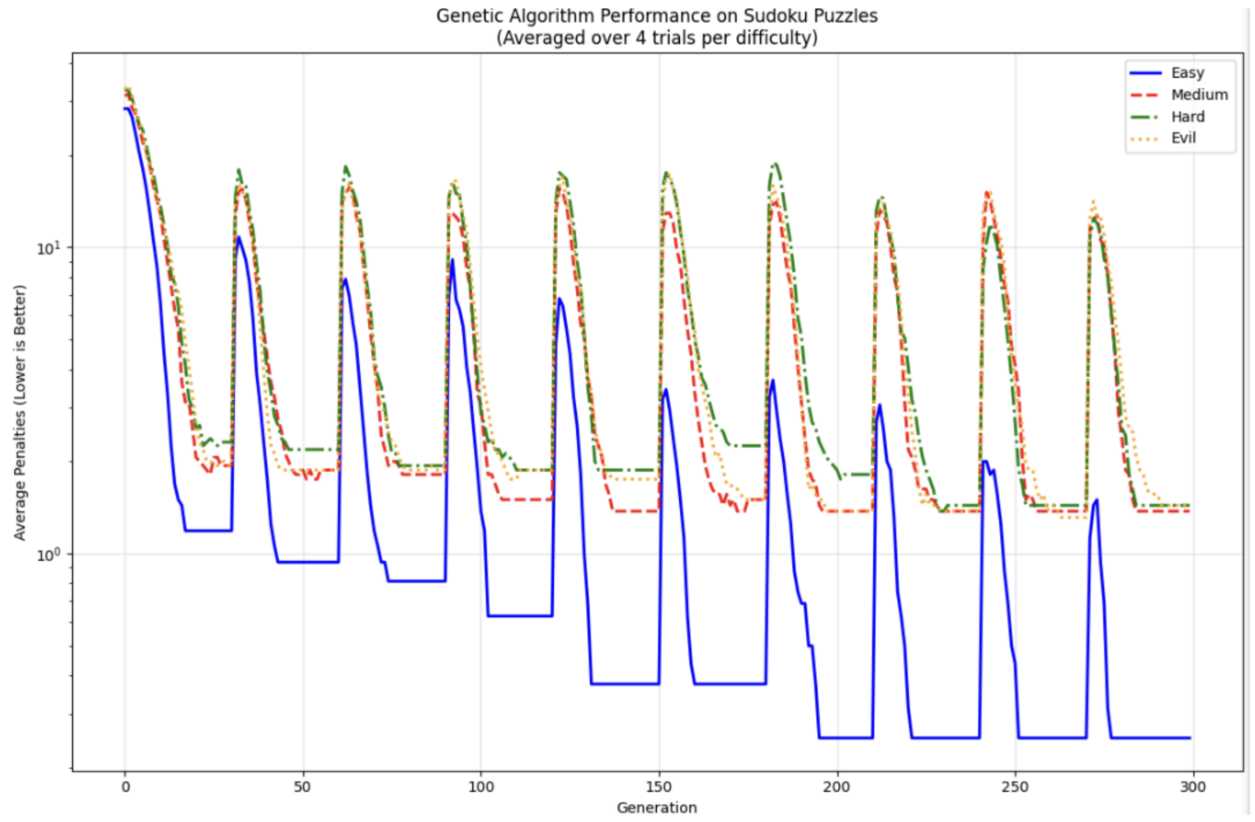
Simulated Annealing



Picture 4

Picture 4: This plot shows the trajectory of the simulated annealing algorithm while attempting to solve a med difficulty puzzle. The decrease in energy shows the algorithm approaching a better solution, and the intermittent restarts are to avoid being stuck at a low temperature. The simulated annealing algorithm is not guaranteed to find a solution so it is very helpful to be able to see the progress it makes even if it does not end on a correct solution. The algorithm is fully solved when the energy is zero, but the reported statistics for this plot do not include the final value of zero so it does not look like it succeeded, and the validator needs to be used to make sure it is not ending before the maximum steps set. This plot is log scaled so it may not be quite as apparent that the energy values can fluctuate much more at a higher temperature which correlates to the algorithm being more willing to make an uphill move and accept more conflict while swapping cells. The final parameters used were max_steps=500000, start_temp=1.4, cooling_rate=0.9998, restarts=11, these can be widely varied depending on the difficulty of problems you are attempting to solve and the speed you want to achieve. Having a cooling rate very close to 1 allows the algorithm to spend a lot of time searching at every temperature level so it will be more robust, and a larger starting temperature allows it to start off more chaotic.

Genetic Algorithm



Picture 5

Above is a plot showing our genetic algorithm's results. The spikes in the plot can be explained by the feature added that initiates a new population every 30 generations. We did this because we noticed that at around 27 generations the output was stagnant and would not improve (as shown by the flat lines at the end of each iteration in the graph) so the idea behind refreshing the initial population is to give it more chances to solve the puzzle. It takes approximately 30 seconds for our algorithm to run through 300 generations at a 1000 population size but it will load faster if it finds the answer. Something interesting about this graph is that the evil difficulty was more consistent with its penalties per iteration than the hard puzzles; however, this is probably an anomaly. Our final hyperparameters for this algorithm were mutation_rate = .3, tournament_size = 4, population_count = 1000, and max_generations = 300.

BEHAVIOR OF THE ALGORITHMS

Observing the behavior of our five sudoku solving algorithms allows us to identify distinct strengths and weaknesses that shape their practical performance across puzzle difficulties.

Backtracking is a straightforward and simple approach. It guarantees a solution if one exists. It does so by exploring all possibilities, but its efficiency suffers for more difficult puzzles. As complexity increased, the number of assignments and solve time increased greatly. Backtracking performed best on easy and medium puzzles, where its simplicity created fast and accurate results.

Forward Checking improves upon backtracking by eliminating values that would lead to conflicts before making assignments. This reduces the number of dead ends and un-needed assignments. In doing so, the algorithm becomes more efficient than simple backtracking. This efficiency becomes very noticeable when solving complex puzzles like "hard" and "evil" as it avoids wrong paths earlier than simple backtracking. For easier puzzles, its performance was similar to backtracking, but its advantage became more pronounced as the difficulty increased.

AC-3 uses constraint propagation to reduce the search space before it tries to solve. It is quick to identify impossible values and altering the search space to simplify the puzzle. Its solve time often lags behind Forward Checking, but its average assignments are very similar to forward checking. AC-3's performed best in the initial stages of reducing the search space before solving where it can reduce complexity before deeper searching.

Simulated Annealing is a randomized optimization method that starts with a random solution and iteratively makes changes. This approach avoids getting stuck in non-optimal solutions by randomly accepting moves that contain more conflicts, especially at the beginning of the run (simulating high temperature) and implementing multiple restarts in order to search large spaces that deterministic methods might have trouble with. It is not complete and due to using random probability to make choices within the algorithm its performance can vary greatly between runs.

Genetic is a stochastic search technique that mimics natural selection. It evolves a group of solutions over generations. This process is slower and less consistent than the simpler backtracking methods and for standard puzzles. This inefficiency is caused by the randomness of evolution and the parameters set for it. Genetics' strong point is the hardest puzzles where simpler algorithms might fail. Refreshing every 30 generations helped improve its chance of success.

SUMMARY

Our project explores five distinct algorithms for solving Sudoku puzzles of varying difficulty. Each algorithm offered unique strategies, giving us a good look at how each of them work. By testing all five across a set of 16 puzzles ranging from easy to evil in difficulty, we were able to observe how each algorithm behaved under increasing complexity.

Backtracking proved to be the most reliable for simple puzzles, with quick and consistent results. Its performance suffered significantly when attempting the hard and evil puzzles. Forward checking improved upon this by closing invalid paths early. This resulted in fewer assignments and faster solve times for the harder puzzles. Ac-3 contributed with its ability to reduce the search space before attempting to solve the puzzle, though its sole time was higher due to the overhead of constraint propagation.

Simulated Annealing can be an extremely robust algorithm even for the most difficult puzzles. It is probabilistic so it is able to search many places that a deterministic method can get stuck in. Tuning the parameters can allow it to spend a very very long time working towards the most optimal solution with great reliability, however it can also waste resources and be inefficient when running like that.

Genetic Algorithm offered an evolutionary approach that came with less consistency and slower speeds than the simpler algorithms on standard puzzles. The genetic algorithm is largely reserved for the instances where traditional algorithms fail. The use of population refresh and parameter tuning helped improve performance and avoid stagnation.

Overall, our testing across 16 Sudoku puzzles revealed that no single algorithm can consistently outperform the rest. Instead, each algorithm has their spot and time to shine, and it's important to understand the optimal times to use each one. The simpler methods are best suited for easy and medium puzzles, while the more advanced techniques like simulated annealing and genetics are better suited for difficult puzzles. Understanding when and where to apply each algorithm is key to solving the Sudoku puzzles efficiently.

REFERENCES

Isaac Walter

- none

Tyler Foster

- none

Robert Phipps

- none