Benjamin Heinze, Braxton McCormack

Ben.c.heinze@gmail.com, braxton.j.mccormack@gmail.com

Group #8

17 September 2023

Project 1 – Search, Constraint Satisfaction, and Sudoku

Abstract----------

This paper presents our group's analysis of using various search algorithms to attempt to find Sudoku solutions. Our project explored the following algorithms, Backtracking, Backtracking with Forward Search, Backtracking with Arc Consistency, Simulated Annealing, and Genetic Algorithms to solve Sudoku puzzles of differing difficulty levels. We hypothesized that Backtracking along with its sub methods would outperform the others in terms of a lower number of decisions made but we expected that Simulated Annealing and Genetic Algorithms would have more unique solutions. Through experimentation we implemented each algorithm on various Sudoku puzzles, ranging from easy to evil difficulty, measuring decisions made and fitness at each decision point. We were able to conclude that Simulated Annealing and Genetic Algorithms take the longest to reach a viable solution, however due to their randomness they were more likely to provide unique solutions on each run. Backtracking, on the other hand, was able to find a conclusive solution to each puzzle much faster, never taking more than ~1,500 cycles to find a solution. The AC-3 and Forward Checking upgrades only improved the time it took for backtracking to find a solution. Overall, our experiments revealed the strengths and weaknesses of various search algorithms and their effectiveness in solving Sudoku puzzles.

Problem Statement and Hypothesis----------

Our objective is to investigate the performance of five different algorithms against the constraint satisfaction problem (CSP) Sudoku. Three of the five use a form of recursive backtracking and the other two use local searching. We expect the backtracking solutions to always guarantee a solution since the only variance is how quickly they accomplish the task. Due to the random nature of local searching, we are not expecting to always come to a solution, however we will show how these algorithms converge towards one. We expect each algorithm's effectiveness will differ depending on the puzzle's difficulty level. There are four categories of difficulty (Easy, Med, Hard, Evil), we expect harder puzzles to take longer than easier ones.

Description of Implemented Algorithms----------

We implemented the following search algorithms:

- Simple Backtracking
    - Simple backtracking is a brute-force algorithm in the context of Sudoku. It starts by filling cells with random values until it reaches a point where it cannot find a valid number for the cell. It then backtracks to a different point and tries a different number. It continues until a solution is found.
- Backtracking with Forward Checking
    - Backtracking with forward checking is an upgraded version of Simple Backtracking. It maintains a list of remaining cells for the unfilled cells and updates the list each time a value is assigned to a cell, and it removes that value from the list. In our case we had 27 lists for each constraint (9 columns, 9 rows, 9 sub cells). When placing a value, it updated the 3 lists affected by the location of the number.
- Backtracking with Arc Consistency

    - Backtracking with arc consistency is also an upgraded version of Simple Backtracking. In our case we check the Arc Consistency of our list of remaining values at the start of our backtracking. We update the relevant lists of available numbers by removing the pre-established immutable values already in the table from each list of values effected by their location.
- Local Search Simulated Annealing with Minimum Conflict Heuristic
    - In the context of Sudoku our Simulated Annealing algorithm keeps track of a board cost (with 0 being a correct solution), representing conflicts. The algorithm then iteratively improves the board by making random moves and gradually decreasing the temperature. As the temperature decreases the board allows more random moves. The algorithm tracks the best solution found and the number of decisions made.
- Local Search using Genetic Algorithm with a penalty function and tournament selection.
    - Our Genetic Algorithm initializes two parent boards populated with random values. The algorithm runs for a set amount of cycles and ech cycle it performs a crossover between the two parent boards, applies mutations, calculates fitness for each child.

Experimental Approach----------

We are measuring the effectiveness of the backtracking algorithms by counting the number of times the algorithm inputs a number into a cell, we will reference this as a decision. The lower the number of decisions, the more effective the algorithm is. Therefore, we expect Simple Backtracking to make the most decisions, Backtracking-forward-checking with less decisions, then backtracking with arc consistency should have the least number of decisions. For the local searching algorithms, we will compare the number of collisions (inconsistencies) against the number of cycles (epochs). The best-case scenario is that the number of collisions is zero meaning the puzzle is solved. That may not be feasible for every algorithm; by measuring the number of collisions against the number of cycles, we can prove that these algorithms converge towards a solution.

To evaluate the performance of our algorithms, we conducted experiments by running each algorithm on every puzzle. Each decision count increment we wrote the value to a csv file along with the fitness allowing us to observe how the fitness value converged towards zero (a solved board).

Results----------

Simple Backtracking:

	Easy	Medium	Hard	Evil
1	298	1354	9670	5954
2	1354	13776	39788	3792
3	812	3022	7898	8750
4	262	2114	6332	140
5	450	26174	1012	4040
Average	635.6	9288	12940	4535.2

Although there is a general trend that the number of decisions increases with difficulty, the average is not linear with the puzzles we were given. The average is harshly skewed by specific outliers. For example, M*ed-P5* had an unusual score of over 26174 decisions, while *Evil-P4* scored under 150 decisions.

Backtracking Forward Checking:

	Easy	Medium	Hard	Evil
1	150	677	4826	2978

2	678	6888	19895	1897
3	407	1511	3950	4376
4	132	1057	3166	71
5	226	13087	507	2021
Average	1593	4644	5678.8	2268.6

In these results we can see that implementing forward checking into our Backtracking algorithm reduces the number of decisions it takes for the algorithm to reach a solution. Each time our backtracking algorithm would place a value it would remove that value from the domain of possible solutions for the future. This helps to limit the amount of time future decisions take to make.

Backtracking Arc-Consistency:

	Easy	Medium	Hard	Evil
1	224	1015	7263	4456
2	1016	10332	29842	2845
3	609	3367	5924	6563
4	197	1586	4749	106
5	338	19631	760	3130
Average	476.8	19231	9707.6	3420

When we implement AC-3 we can see that it is also an improvement apon normal Backtracking. You may notice that the decisions made here are still much higher than with forward checking. That is because with AC-3 we only reduce the domain for every number of values already in a puzzle. In every puzzle this is less than the number of blank spaces so that means the domain of responses will be trimmed much less in AC-3 than in Forward Checking.

Local Search Simulated Annealing:

Fitness value of the Simulated Annealing boards after 10,000 cycles:

	Easy	Medium	Hard	Evil
1	5	9	7	10
2	7	6	6	8
3	13	10	7	7
4	4	7	9	8
5	9	8	7	7
Average	7.6	8	7.2	8

Here we can see that while Simulated Annealing is unable to reach a solution in 10,000 cycles the fitness values (0 is a perfect score) do converge towards 0. This algorithm has an extremely similar performance no matter the difficulty the puzzle is given to it with the average for each puzzle all being within .8 of each other.

Local Search Genetic Algorithm:

The goal of the genetic algorithm is to simulate generational adaptation found in nature. My specific process consisted of two parents (a sudoku board with randomly filled cells). We randomly picked a cell in the middle of the board, then swapped the top and bottom pieces between the two parents. Then a fitness test evaluated the parents and the two new offsprings. Whichever two of the four had the lowest number of inconsistencies was chosen for the next generation. Once the new parents were

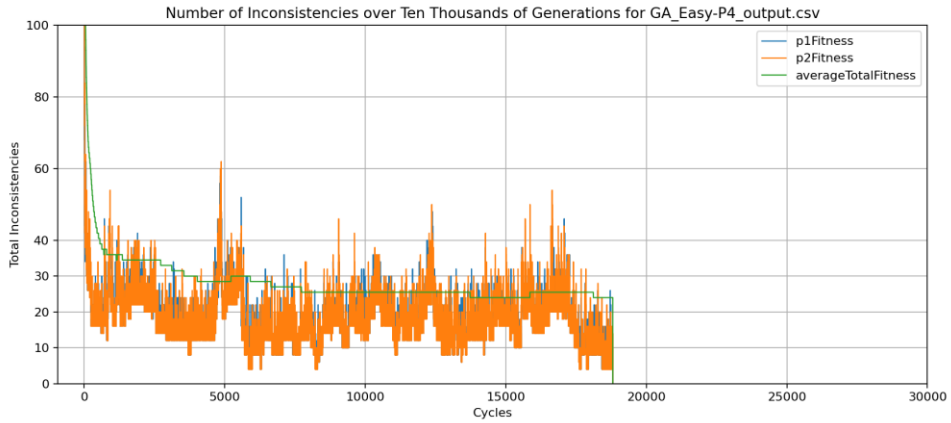
assigned, we mutated each parent and recorded their fitnesses again. The goal is to have the number of inconsistencies converge towards zero.

nCycles	Threshold	upperBound	highProbability	lowProbability
30000	1-3	1000	900	1-5

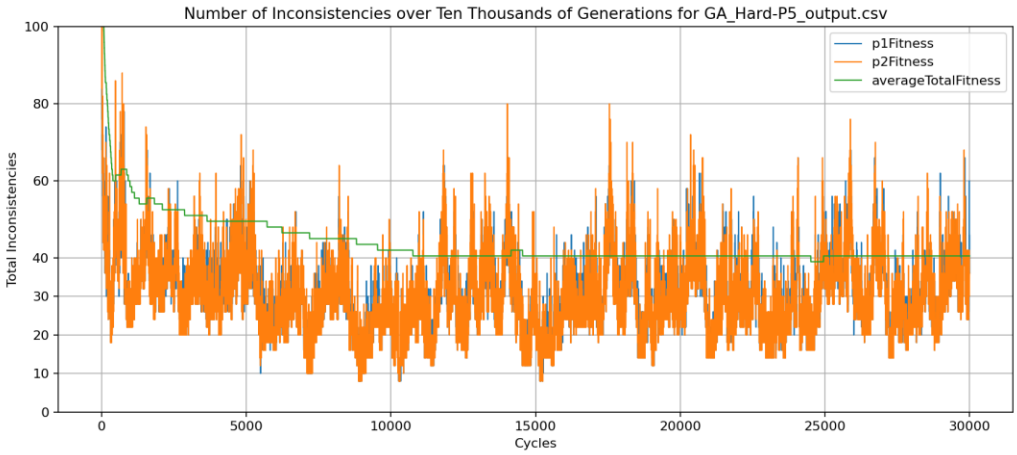
- nCycles: How many times the GA cycle performed one loop, which consisted of performing crossover, then mutation, then judging fitness.
- Threshold: This dictated whether the number of inconsistencies a single cell had was worth a high probability to mutate or low probability to mutate. If the number of inconsistencies was less than the threshold, it would have a low probability and vice versa.
- UpperBound: When generating probabilities for mutations, upperBound represents the possible range for the randomly generated number.
- HighProbability: The chances a cell with a high number of inconsistencies mutates into a new number
- LowProbability: The chances a cell with a small number of inconsistencies mutates into a new number.

When determining this parameter, we observed runs from 100 cycles to 100,000 cycles and concluded the time it takes to run is not worth lowering the overall average. In the *Easy-P1* graph it is apparent that it locks around four inconsistencies. After that, solving the puzzle is pure luck and it would be more efficient to use backtracking to fix the rest of the puzzle.

It quickly became apparent that the mutation function determines the degree and speed of convergence. Our GA algorithm can solve puzzles as shown in the first graph below inconsistently. The data from comparing easy graphs to difficult graphs suggest that the difficulty of the puzzles influences where the average converges.



Harder puzzles will converge at slightly higher numbers. For example, *Easy-P4* has a total average that began to flatten around 25 while the second graph, *Hard-P5*, began to converge around 40.



We hypothesis the premature converge is caused by the high selection pressure imposed from the small number of inconsistencies. To potentially negate this in the future, we could see if changing

Discussion----------

Simple Backtracking: This algorithm was the most consistent in terms of finding solutions. It was able to solve all puzzles but on occasion we would run into an outlier in the number decisions it would take to solve a puzzle.

Backtracking with Forward Checking: This upgrade to Simple Backtracking showed improvement. By maintaining a list of values for unfilled cells and updating that list each time we placed a number increment the number of decisions. The algorithm efficiently updated these lists as values were assigned to cells, which restricted the domain of potential solutions and led to the solution being found much sooner. In conclusion this upgrade is a noticeable improvement to Simple Backtracking.

Backtracking with Arc Consistency: Arc Consistency is another upgrade to simple backtracking. While not as effective as our Forward Checking it still performs better than Simple Backtracking. Similar to Forward Checking, this algorithm keeps lists of possible values for unfilled cells, and it updates these lists before running the Backtracking Algorithm with the values and locations of the immutable values in the puzzle given.

Simulated Annealing: Simulated Annealing relied on local search and randomness. It proved to be consistent across puzzles of all difficulty levels. This algorithm took an exceedingly long time to reach a solution and in under 10,000 cycles it was unable to find a solution at all. The fitness values of the generated solutions often would approach 0 (a solved board) with most values being less than 10 in 10,000 cycles.

Genetic Algorithm: Due to the high selection pressure when the puzzle's inconsistencies were low, we were not able to solve it purely with the genetic algorithm. Combining the GA with the backtracking algorithms would yield the best results, though finding better parameters for the mutation function would help as well.

Summary----------

In summary, our study compared the effectiveness of different search algorithms in solving Sudoku puzzles. Simple Backtracking consistently found solutions within a reasonable number of decisions. Backtracking with forward checking and Arc Consistency reduced decisions counts by maintaining lists of values. Simulated Annealing, while not providing a solution in under 100,000 cycles, provided more unique solutions, but they did not have a fitness score of 0, only close to 0. Genetic

Algorithms had similar performance to Simulated Annealing but was more likely to find a solution, though those solutions are luck-based.

References----------

Levine, John. Constraint Satisfaction: The AC-3 Algorithm. YouTube, 14 Dec. 2019, https://youtu.be/4cCS8rrYT14?si=MHB8XkCbzfEuxBrw. Accessed 1 Sept. 2023.

Russell, Stuart J., et al. "Chapter 4 Search in Complex Environments, Chapter 6 Constraint Satisfaction Problems." Artificial Intelligence: A Modern Approach, Fourth Edition, Fourth ed., Pearson Education, Harlow, 2022.