Final Report Project 1

Applying Five Search Methods to a Constraint Satisfaction Problem: Sudoku

Group 26: Nitasha Fazal & Angelo Porcella

Abstract

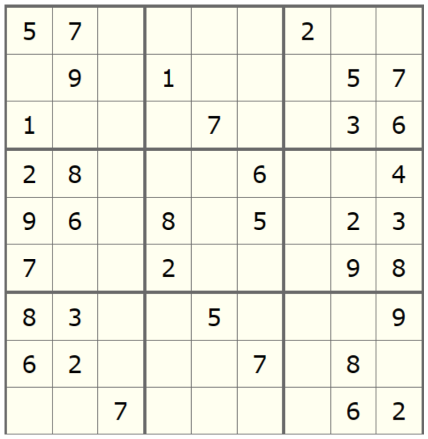
This research aimed to find optimal search methods to generate valid solutions for a Sudoku puzzle, a constraint satisfaction problem. The five algorithms implemented and tested were the following: Simple Backtracking (BT), Backtracking with Forward Checking (FC), Backtracking with Arc Consistency (AC), Local Search with Simulated Annealing and a Minimum Conflict Heuristic (SA), and finally, Local Search using a Genetic Algorithm with a Penalty Function and Tournament Selection (GA). Algorithm efficiency was determined by measuring the number of evaluations being made, the success rate, and the number of cell replacements performed. Testing determined, of the deterministic methods, AC had the best performance while finding a solution overall. The stochastic methods were unfortunately not able to consistently find a valid solution, but of the two, GA had better measured performance overall, while SA was closer to finding a solution.

Problem Statement

Problem: Given a constraint satisfaction problem (CSP) there are many ways to approach and produce a valid solution. In the case of the Sudoku puzzle, the problem presented is to figure out which approach is best for puzzles of varying difficulty. These approaches differ in complexity, processing time, and overall effectiveness based upon how hard each problem is.

Background: In the context of a Sudoku puzzle (*figure 1*), the constraints consist of having the digits one through nine represented only a single time in each domain. This results in a valid 9 x 9 puzzle where each digit is only represented once in each row, column, and 3 x 3 sub grid.

The algorithms implemented fall into two categories, deterministic and stochastic and will be explained in greater detail below. The three deterministic algorithms involve backtracking to build an incremental solution. The two stochastic algorithms involve local search.



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

Figure 1: Example of an empty puzzle.

Objective: The purpose of this research is to observe the performance of five different algorithms and determine the overall effectiveness of each, utilizing established criteria. These include the number of cells placed/replaced, the number of evaluations performed, and the success rate.

Hypothesis: It is the belief of this research group that backtracking methods will perform better on easier puzzles, whereas the local search algorithms will be more efficient at solving the hardest puzzles presented. In particular, the most advanced backtracking algorithm involving Arc Consistency (AC) will likely be the optimal algorithm for easy to medium puzzles, and the Genetic Algorithm (GA) will likely be more consistent at finding a valid solution on the hardest problems when compared to Simulated Annealing.

Algorithms Used

Simple Backtracking: Simple Backtracking (BT) is a deterministic recursive algorithm that builds a solution by placing digits in an incremental fashion starting in the top left corner of the puzzle. After each placement a verification method is called to determine if any conflicts arise. If a conflict is found, a "back track" occurs wherein the algorithm returns to the point where a placed digit resulted in an invalid solution. The number placed is increased by one and solving continues in the same fashion until a solution is found. If the first cell that takes a digit reaches the final number (nine) and a backtrack is triggered, the solution doesn't exist for the puzzle in question. This is considered a 'brute force' approach as all possible solutions are tried until a valid completed puzzle is formed. This algorithm is deterministic, in that it will perform the same on a given puzzle regardless of how many times it is run.

Backtracking with Forward Checking: Backtracking with Forward Checking (FC) is a deterministic recursive algorithm that builds upon BT resulting in a more efficient search for the solution. In addition to the qualities of BT, FC incorporates a "Forward Check". This involves considering the domain of each cell and tracks the possible digits that can be placed. When the BT portion of the algorithm places a value in a cell, that value is removed from its domain. That is, every cell in the row, column, and 3 x 3 sub grid shared by the cell filled, has the placed value removed from its list of possible values. A forward check is then performed, where each cell in the puzzle has its possible value list checked to determine if a cell has no possible values in this branch of the recursive loop. If such a cell is found, a back track is immediately triggered, in theory discovering an invalid solution before simple backtracking would, reducing the need for repeated verification.

Backtracking with Arc Consistency: Backtracking with Arc Consistency (AC) builds upon the efficiency of FC by adding in a faster method of determining if a particular branch is going to result in an invalid solution. This involves detecting when a lone digit possibility is shared between two or more cells in a domain. Rather than waiting for the possibilities to be exhausted, a backtrack is triggered when this scenario is found. Further reducing the need for verification over the FC and BT algorithms.

Local Search using Simulated Annealing with a Minimum Conflict Heuristic: This algorithm (referred to as SA for future reference) emulates the tempering of forged metal to strengthen the final product. A state is produced by randomly filling in an empty puzzle then running a cost function to determine the total number of conflicts in the state. To improve performance, a constraint was placed this random generation, where the filled puzzle could not have any conflicts in each 3 x 3 sub grid. This ensured a baseline level of conflict mitigation. The algorithm starts at a "high temperature" which is determined by a function that calculates the standard deviation for ten new states and returns the average standard deviation. Starting at this temperature, the algorithm is looped through with a tunable cooling rate of 1%. This continues until the terminating temperature of 0.01 is reached. In each loop, a new state is generated and compared to the current. If the cost of the new state is less cost than the current state, the new state is likely to be chosen. If the new state is not less than the current, the current state remains. However, there is a chance the worse state is still chosen at high temperatures. As the temperature cools, the algorithm gets greedier and the likelihood of selecting the lower cost state increases. When the terminating temperature is eventually reached, the optimal solution is the current state selected.

Local Search using a Genetic Algorithm with a Penalty Function and Tournament Selection: The last algorithm implemented is a Genetic Algorithm with a Penalty Function and Tournament selection (GA). This stochastic search method begins by generating a population size of 200 individuals. These individuals are created by randomly filling in the selected puzzle. To improve performance, the same constraint for SA was placed on individual generation. After population generation, tournament selection proceeds where ten members are chosen. The members are paired off and whichever individual has a lower total cost (i.e., a more "fit" specimen) moves on to the next round. When two of the most fit individuals are selected, crossing over and then mutation occurs between them. Crossing over is swapping a portion of each board. In this implementation, the first three columns on each board were chosen to swap. Mutation involves changing the population slightly to make them more fit. This process continues until a child is generated with no conflicts.

Experimental Approach

The three measurements taken to determine the effectiveness of the algorithms were the number of evaluations performed (recording the number of times the state of the puzzle is evaluated), the number of replacements (swapping or placing a value in a cell), and cost (how valid the final solution is). For the deterministic algorithms, since each puzzle will be solved in the same way no matter how many times it is run, these only needed to be run once on each puzzle. For the stochastic algorithms, each was run on every puzzle ten times and the resulting data was recorded.

Data analysis was conducted via a two-way ANOVA. This was to determine if there was a statistically significant difference between the performance of the algorithms, and the difference in performance puzzle difficulty presented.

Results

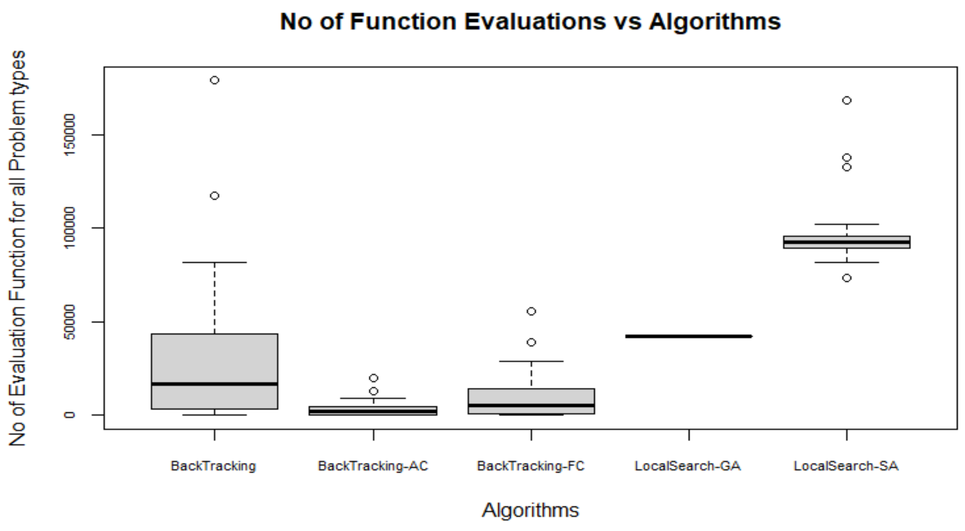


Figure 2: Box and Whisker plot for the number of evaluations performed for each method.

Figure 2 displays the number of function evaluations performed across each method. The Y axis represents the total number of board verifications performed while the X axis shows each algorithm. This results in a box and whisker plot that visualizes the span of performance across all puzzles.

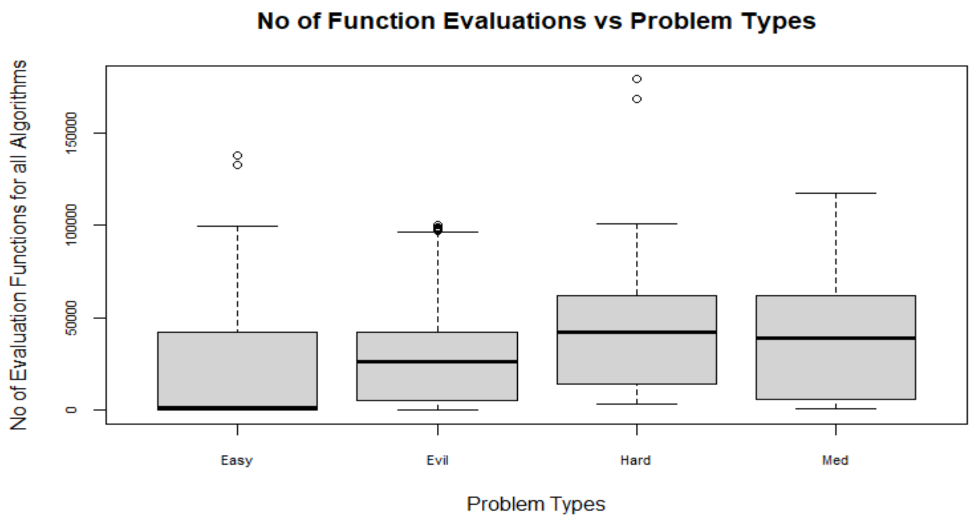


Figure 3: Number of Evaluations depending on problem types.

Figure 3 displays the number of times verification needed to be performed across all algorithms. The Y axis displays the number of verification steps, and the X axis displays the different problem types. This graph is a box and whisker plot that visualizes the span of performance based upon the difficulty of the puzzle across all algorithms.

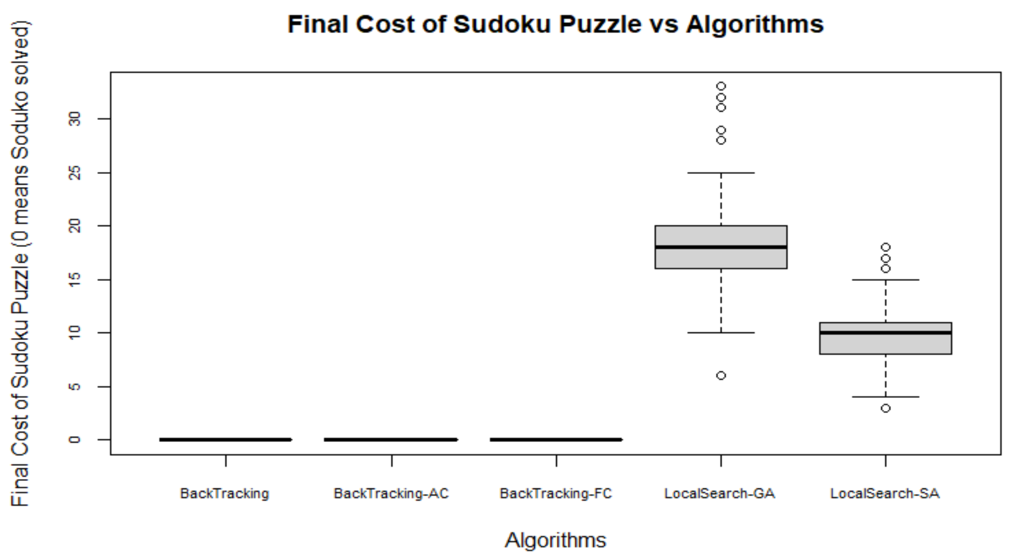
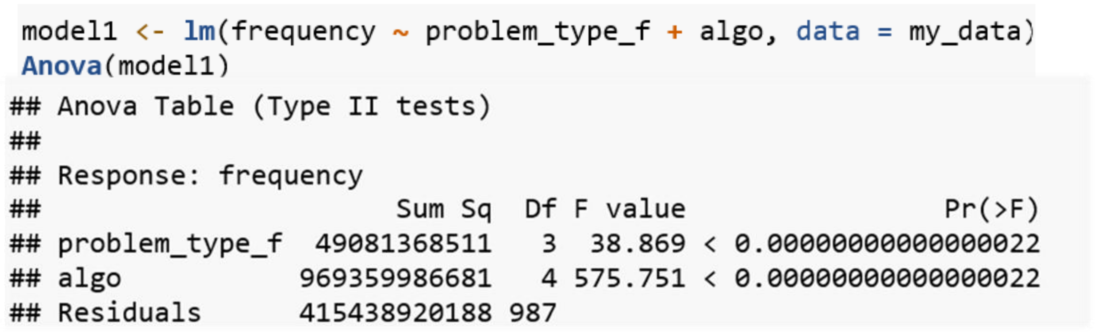


Figure 4: Cost for each method (number of conflicts) for each algorithm when run on all puzzles.

Figure 4 displays the cost of the solution found across all puzzles, a cost of zero indicates that a final verified solution was found. The Y axis displays the cost, and the X axis shows each algorithm. This is a box and whisker plot that visualizes the success rate of each algorithm across all puzzles.



```
model1 <- lm(frequency ~ problem_type_f + algo, data = my_data)
Anova(model1)
## Anova Table (Type II tests)
##
## Response: frequency
##                       Sum Sq  Df F value                  Pr(>F)
## problem_type_f   49081368511   3  38.869 < 0.00000000000000022
## algo            969359986681   4 575.751 < 0.00000000000000022
## Residuals       415438920188 987
```

Figure 5: Two Way ANOVA results.

Figure 5 shows the two-way ANOVA results for each algorithm and problem type versus the function evaluation (the total number of verification steps). A P value of < 0.001 and an F value of 38.869 are presented for problem type versus the number of verifications. A P value of < 0.001 and an F value of 575.751 are presented for algorithm used versus the number of verifications.

Discussion

Conclusion: According to the data presented, we can conclude that the deterministic algorithms performed best with our implementations. Namely, AC held the best performance across the board. This result was expected and falls in line with the hypothesis stated.

For the stochastic methods, between GA and SA, GA boasted better performance, while SA provided a more accurate solution consistently. This goes against the initial hypothesis claimed.

Unfortunately, the consistency of producing low-cost solutions was not at the desired level. However, the data shows that cost improvements did still occur.

The results of the two-way ANOVA suggest that the problem difficulty, and the algorithm used are significant variables that affect performance. The extremely low P values in both tests indicate a statistically significant difference in performance across puzzle type, and algorithm.

Difficulties: Throughout this study, there were challenges that had to be overcome. One of which was the game loader class created issues when importing puzzles. Invisible characters placed at the start of each file that were not evident when printing/viewing said file caused some bugs that were difficult to trace.

In addition, the implementation of the stochastic methods came with difficulties, as a consistent valid solution was extremely difficult to come by. This could be due to the implementation itself, or simply not finding the correct tuning for the algorithms.

Changes from Design Document: One of the most obvious changes from the design document to the final experiment is the omission of a GUI for the experiment. Time constraints made it difficult to prioritize this less than necessary feature when simply collecting data was a top priority.

Possible Future Questions: It would be interesting to see how mixing and matching certain algorithms into a more complex method would impact performance. For example, if SA also incorporated portions from a GA, would this increase its effectiveness? Or if a portion of the puzzle was solved via SA, and then AC takes over to finish up the puzzle. There are many more possibilities that could be tried even with these few algorithms. As circumstances may dictate which method is the most efficient for a problem, perhaps a mix of several algorithms would result in an efficient general solution.

Summary

In summary, the desired results for this experiment were not achieved, although meaningful data was collected. The data shows the algorithms working and solving the presented problems, but the stochastic methods (GA and SA) namely, simply showed an improved answer rather than a completely valid one. Of the deterministic search methods, AC boasted the best performance. This was expected, as it is the most sophisticated implementation of a back tracking method performed on this CSP.

References

“Blank Sudoku Grid with Features - Play Your Own Sudoku Puzzles Online.” *Sudoku9x9.com*, sudoku9x9.com/sudoku_blank_grid_9x9.php. Accessed 21 Sept. 2023.

Russell, Stuart, et al. Artificial Intelligence: A Modern Approach. 4th ed., Pearson, 2022.