CIS 4930 – Introduction to Artificial Intelligence

Group 1:

Osniel Quintana

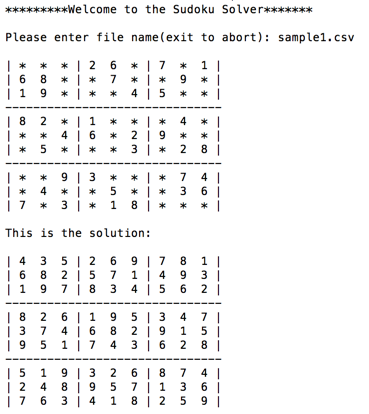
Brandon Nunez

Alec Mandell

Sudoku Solver Report

The popular game, Sudoku, has been around for many years and its popularity has grown greatly over the years. The growing popularity has led to many published books with hundreds of the puzzles, being placed into daily newspapers, and the development of many apps with the puzzles. For those who are not familiar with Sudoku puzzles, a puzzle consists of a 9-by-9 square starting off with a select few numbers present that result in exactly one solution. The puzzle is considered solved when every block within the square are filled with a number ranging from 1 - 9 and do not break any of the three rules. Rule #1: For every row within the puzzle there is no repeat numbers of 1 - 9. Rule #2: For every column within the puzzle there is no repeat numbers of 1 - 9. Rule #3: For the specified 3-by-3 blocks within the puzzle there is no repeat numbers of 1 - 9. The main purpose of our project is to implement an algorithm, which will read in a CSV file with the matching setup of some Sudoku puzzle and output the solution of that particular Sudoku puzzle within a reasonable amount of time.

The way we decided to tackle this problem was utilizing the CSP arc consistency to implement a backtracking search. CSP standing for constraint satisfaction problem, implies that the problem revolves around some sort of constraints, which in the case of a Sudoku puzzle are the rules that are to be followed in order for the solution to be considered complete. The main idea behind a CSP is to eliminate values that could not be apart of the solution. Once a value is deemed not suitable for the solution, how will the program continue to run? This is where backtracking comes into play. The purpose of backtracking is when a value is deemed not suitable for the solution, the algorithm iterates backward to a state where it is able to continue searching and placing values until the puzzle is solved.

The basic way to test that the algorithm was running correctly was by comparing the inputted CSV files with their actual solutions and the outputted solution from the algorithm. As important as it was to assure the algorithm completed the task, the next (and probably most important) was determining how long it took the algorithm to solve the puzzle. The time complexity for the Sudoku puzzle is O(bd) where b is the number of possibilities per square and d is the number of empty squares. Visually judging based off the completion time, the algorithm works exceptionally well and produces the correct solution for all our test CSV files. In order to get a better understand of how the algorithm was working behind the scenes, we decided to create a static variable which would count each time the backtracking was called to determine how well the heuristics were driving the algorithm in the right direction and to give us a rough idea of how many moves it took before solving the puzzle. 

In conclusion, we have determined the algorithm in which we implemented to be optimal and complete assuming there is a solution possible for the specified Sudoku puzzle. There was shocking results when analyzing the difference of moves taken to complete the puzzle for each experiment. The first run depicted above took 291 moves until finding the solution for that puzzle, but then another test run on a different puzzle took 16,248 moves until finding the solution. For a computer nowadays making that many moves occurs in a matter of nanoseconds, but the results are shocking because of the great difference between the amount of moves it took for the algorithm to solve each puzzle. The only reason for this difference is probably due to the way the puzzles were set up. Looking closely at the puzzles, the main difference between them from the start is that the first one(one with the less moves) had all the numbers scattered fairly nicely which was helpful to lead the search in the right direction, while the other puzzles numbers were more skewed in different regions and the middle 3-by-3 area was completely empty, so during the search there was probably many points where the search would have to restart from some point. That happens to be the only surprising data recorded, but the great news is it did not cause any issues during the search and the algorithm was able to produce a solution within a very short period of time.