**SUDOKU**

Ninad Khalate G00948398

Mukesh Kumar G00973428

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Type  Algorithm | Easy | | | Medium | | | Hard | | |
| fastest | slowest | avg | fastest | slowest | avg | fastest | slowest | avg |
| BT | 11 | 21 | 15 | 8 | 64 | 32 | 19 | 42 | 28 |
| BT + FC | 7 | 802 | 178 | 175 | 3622 | 1389 | 28 | 1670 | 590 |
| BT +AC3 | 40 | 374 | 110 | 312 | 4161 | 2082 | 89 | 5924 | 1495 |
| Heuristic | 24 | 78 | 38 | 46 | 105 | 72 | 52 | 161 | 88 |
| SA | 15 | 128 | 72 | 54 | 94 | 78 | 57 | 91 | 71 |

**Project Report:**

Summary Table:

Runtime comparison plot for all algorithms for the 3 categories:

**Interpretation:**

After running these 15 puzzles with the given 5 algorithms we came to a conclusion that every algorithm has its own positives given a particular puzzle. Theoretically Backtracking with Forward checking and backtracking with AC3 should take lesser time than the normal backtracking algorithm but here while implementing these algorithm in java we understood that the runtime for a given algorithm will depend on more than just how the algorithm works, the data structure we used and the memory operations done contributed heavily on the time taken by the algorithm to solve the given Sudoku configuration.

When it comes to deciding which algorithm is best for solving Sudoku it’s a difficult decision to make as the runtime for just these 5 variants of each category were very diverse. So it is really not fair to decide if the algorithm is best just by testing it on 15 Sudoku configurations. But still we get a rough idea about which one will be ideal. According to our tests we found out that heuristic based algorithm which we chose and simulated annealing were more apt choice in most of the cases. The MRV heuristic and simulated annealing both had an almost constant runtime for all the 15 puzzles. Thus they both were very consistent. So we came to a conclusion that heuristic(MRV) based algorithm would be best for the Sudoku problem in general

**Project Summary:**

For Forward checking add a forward check while backtracking so as to know the dead end earlier. Even though theoretically this should run faster than the normal backtracking this takes longer time here due to the extra memory accesses it takes.

The simulated annealing program is designed in a way such that it resets the board if it is stuck in a local minima where the time to write out the misplaced values in the cell takes longer than having another go in solving the puzzle. After multiple runs on a test board we have come to a conclusion where the board is reset to the initial state after it exceeds any more than 15,000 iterations remaining on the same score. The temperature for the Simulated Annealing is set to 0.8 and is set to update by multiplying the temperature with values to make the resulting temperature value lower than before. The number of iterations taken by the simulated annealing is much higher compared to the other 4 algorithms implemented in this project as this algorithm involved the execution of multiple redundant steps in searching for the cells. This in fact has a major impact on the runtime of the algorithm making its execution time very waivered as well.

Implementing the Minimum Remaining Value heuristics on the Sudoku board using backtracking is a very smart way to fill cells in each row. It definitely is a smarter solution to fill in the rows which has the least number of unfilled values to the row which has the most. The Backtracking algorithm is used to solve the board but with a simple addition to the existing code would make the program much smarter than it was. With the use of a couple of loops to determine the zero count in each row, the backtracking algorithm would handle the rest of the part in filling out the rows. The Heuristic program used HashMap as the data structure to check the validity of the board. Since, this data structure is different from the one we used in solving the backtracking, comparing the runtime and performance of the two would be unfair due to the advantages and disadvantages provided in the structuring of the program.

Given all this our backtracking runs quicker than rest because it literally has very less memory accesses and just goes on putting values until any constrain is violated in which case it simply backtracks.

**Challenges and Improvements:**

The runtime of a specific input board may vary depending on the path the algorithm takes. Since the algorithm involves the use random numbers from selecting the two cells and deciding if whether the new board should be used even though its score may in no way better than the one before, the time taken to solve the board varies to a great extent. Another problem faced while implementing any of the above algorithms was running out of memory. Frequent Stack Overflow errors were observed due to the creating of new variables in each recursive call. A solution we found was to use looping structures instead of recursive calls on the function as the creation of new objects in each call is said to be avoided and thus making it consume less memory.

Choosing a data structure for AC3 was a big challenge. The arcs needed to be stored in a set of pairs. Also a arc could not be put in the queue twice but here as the arcs were object we specifically needed to overwrite the equals method to get rid of the duplicates.

An improvement on the way simulated annealing algorithm works would be include ridged programming conditions similar to the one we implemented. The algorithm as it is would eventually find the solution to the board without running out of memory, but the time taken for the random search to find both the misplaced cells and rectifying it would be a time consuming process. If excluding this new reset statement we included out program would take around 90 seconds or more to solve the board. No further improvement can be found with Minimum remaining value heuristics as it in itself is an improved version of the backtracking algorithm. Though, there are other heuristics which provide better runtime when paired with backtracking, MRV cannot be improved further comparing with its own benchmarks.

Category : EASY

Easy 1

|  |  |
| --- | --- |
| **Algorithm** | **Times (ms)** |
| Backtracking | 21 |
| Backtracking + Forward Checking | 802 |
| Backtracking + AC3 | 374 |
| Heuristic MRV | 35 |
| Simulated Annealing | 128 |

Easy 2

|  |  |
| --- | --- |
| **Algorithm** | **Times (ms)** |
| Backtracking | 13 |
| Backtracking + Forward Checking | 41 |
| Backtracking + AC3 | 43 |
| Heuristic MRV | 78 |
| Simulated Annealing | 106 |

|  |  |
| --- | --- |
| **Algorithm** | **Times (ms)** |
| Backtracking | 12 |
| Backtracking + Forward Checking | 7 |
| Backtracking + AC3 | 40 |
| Heuristic MRV | 29 |
| Simulated Annealing | 17 |

Easy 3

|  |  |
| --- | --- |
| **Algorithm** | **Times (ms)** |
| Backtracking | 20 |
| Backtracking + Forward Checking | 31 |
| Backtracking + AC3 | 50 |
| Heuristic MRV | 26 |
| Simulated Annealing | 92 |

Easy 4

|  |  |
| --- | --- |
| **Algorithm** | **Times (ms)** |
| Backtracking | 11 |
| Backtracking + Forward Checking | 13 |
| Backtracking + AC3 | 45 |
| Heuristic MRV | 24 |
| Simulated Annealing | 15 |

Easy 5

Category : MEDIUM

Medium 1

|  |  |
| --- | --- |
| **Algorithm** | **Times (ms)** |
| Backtracking | 16 |
| Backtracking + Forward Checking | 419 |
| Backtracking + AC3 | 345 |
| Heuristic MRV | 55 |
| Simulated Annealing | 90 |

Medium 2

|  |  |
| --- | --- |
| **Algorithm** | **Times (ms)** |
| Backtracking | 51 |
| Backtracking + Forward Checking | 2117 |
| Backtracking + AC3 | 3749 |
| Heuristic MRV | 98 |
| Simulated Annealing | 61 |

|  |  |
| --- | --- |
| **Algorithm** | **Times (ms)** |
| Backtracking | 8 |
| Backtracking + Forward Checking | 175 |
| Backtracking + AC3 | 312 |
| Heuristic MRV | 46 |
| Simulated Annealing | 54 |

Medium 3

|  |  |
| --- | --- |
| **Algorithm** | **Times (ms)** |
| Backtracking | 64 |
| Backtracking + Forward Checking | 3622 |
| Backtracking + AC3 | 4161 |
| Heuristic MRV | 105 |
| Simulated Annealing | 92 |

Medium 4

|  |  |
| --- | --- |
| **Algorithm** | **Times (ms)** |
| Backtracking | 21 |
| Backtracking + Forward Checking | 612 |
| Backtracking + AC3 | 1845 |
| Heuristic MRV | 55 |
| Simulated Annealing | 94 |

Medium 5

Category : HARD

Hard 1

|  |  |
| --- | --- |
| **Algorithm** | **Times (ms)** |
| Backtracking | 33 |
| Backtracking + Forward Checking | 844 |
| Backtracking + AC3 | 391 |
| Heuristic MRV | 67 |
| Simulated Annealing | 91 |

Hard 2

|  |  |
| --- | --- |
| **Algorithm** | **Times (ms)** |
| Backtracking | 42 |
| Backtracking + Forward Checking | 1670 |
| Backtracking + AC3 | 5924 |
| Heuristic MRV | 96 |
| Simulated Annealing | 63 |

|  |  |
| --- | --- |
| **Algorithm** | **Times (ms)** |
| Backtracking | 25 |
| Backtracking + Forward Checking | 47 |
| Backtracking + AC3 | 225 |
| Heuristic MRV | 52 |
| Simulated Annealing | 84 |

Hard 3

|  |  |
| --- | --- |
| **Algorithm** | **Times (ms)** |
| Backtracking | 19 |
| Backtracking + Forward Checking | 28 |
| Backtracking + AC3 | 89 |
| Heuristic MRV | 66 |
| Simulated Annealing | 57 |

Hard 4

|  |  |
| --- | --- |
| **Algorithm** | **Times (ms)** |
| Backtracking | 23 |
| Backtracking + Forward Checking | 360 |
| Backtracking + AC3 | 846 |
| Heuristic MRV | 161 |
| Simulated Annealing | 62 |

Hard 5