**CS-171 Final AI Report**

**Team name: Sudokask**

**Member #1([Name&NetID]): Raymond Tu; tur2 Member #2: Mark Sulimov; msulimov**

**I. In about 1/2 page of text, describe what you did to make your Final AI agent “smart.”**

Our Final AI algorithm used several search heuristics and pruning. We first used arc consistency and forward checking on assignments, which would remove values from the domain of their neighbors if a variable is assigned, and if a variable has only one value in its domain, then we would assign that value and call forward checking on that variable. If there are no values left, then we don't have a valid solution and must backtrack. We made this algorithm more efficient by adding an additional optional keyword parameter, last\_assigned\_vars, so we would only need to search the neighbors of one variable. We used several additional pruning methods and heuristics in addition to forward checking. We implemented hidden pair pruning, which is when if a pair of values occur in exactly two neighbor variables, then we can eliminate all of the other values in those two neighbor variables' domains. We also implemented naked pair pruning, which is when if a pair of values is alone in two neighbor variables, then these pair of values can be removed from all other neighbors' domains. We then implemented Norvig's Check, which is when if a constraint has only one possible place for a value, then we assign the only variable with that value.

We also utilized variable and value ordering heuristics. For our variable ordering heuristic, we found the smallest domain variable, then used minimum degree as a tiebreaker, and then the presence of the highest frequency assigned value in the domain of a variable as the second tie breaker. For our value ordering heuristic, we used the heuristic of most frequently assigned value. We found that these heuristics combined with our various pruning methods were more stable than MAD with LCV, in that they were able to solve more boards, although sometimes resulting in more backtracks than MAD with LCV.

These heuristics greatly improved the number of boards solved, and greatly decreased the amount of backtracks by orders of magnitude compared to just a simple backtracking search.

**II. In about 1/4 page of text, describe problems you encountered and how you solved them.**

We encountered several problems with our AI. The first problem we encountered was not quite understanding how the trail worked in Python, so we were unable to backtrack correctly as we did not restore the variable's domain back to what it was before we did forward checking. We solved this by looking more closely at the Variable, Domain, BT Solver, and Trail classes, to better understand how the code works. Another problem we encountered was being unable to zip our project on openlab, since some computers on openlab did not have the zip module installed. We solved this problem by connecting to circinus-1 (via ssh), where there was a zip module installed. The third problem we encountered was an error in the implementation of Norvig. We erroneously tried to count the number of assigned variables on the board and just assigned a value that had a count of p \* q – 1. We ran our incorrect Norvig's checks on several boards, and realized that it was not any better than forward checking. We then rechecked Norvig's algorithm in the slides, and reimplemented it to correctly look through all constraints.

**III. In about 1/4 page of text, provide suggestions for improving the intelligence/performance of your agent.**

The performance of our agent could be improved by adding more pruning/heuristics. Examples of heuristics we could further implement are X-Wing, where if there are two lines, each having the same two relative positions for a number, then we can prune values off of the domain of these variables, Swordfish, which is similar to X-Wing except we look at three rows at once. The more heuristics/pruning we implement, the closer we can get to an ideal linear complexity increase for larger boards. We could also explore different combinations of variable and value selection heuristics to see which ones are the best for different sized boards and then use the best variable and value heuristics for each size of board. The performance of our agent could also be improved by porting it from Python to C++, to have an improvement of about 100x magnitude in running time due to C++ being a compiled language and Python being an interpreted language, which might help our agent to solve more complex boards. We could also optimize the memory usage and time complexity of our various functions and heuristics, which could improve the performance and efficiency of our AI.

**IV. Minimal AI Performance:**

|  |  |  |  |
| --- | --- | --- | --- |
| Board Size | Sample Size | Score (Average backtracks) | Worlds Complete |
| 9x9 | 100 | 42.52 backtracks | 100 |
| 12x12 | 100 | 6849.11 backtracks | 99 |
| 16x16 | 100 | 80646 backtracks | 90 |
| 25x25 | 100 | N/A | 0 |
| Total Summary | 400 | 29,179 backtracks | 289/400 |

**V. Tournament AI Performance:**

|  |  |  |  |
| --- | --- | --- | --- |
| Board Size | Sample Size | Score | Worlds Complete |
| 9x9 | 100 boards | 0.24 backtracks | 100 |
| 12x12 | 100 | 0.2 backtracks | 100 |
| 16x16 | 100 | 0.73 backtracks | 100 |
| 25x25 | 100 | 202.13 backtracks | 100 |
| Total Summary | 400 | 50.825 backtracks | 400/400 |