242 HW2 sudoku report

jiamu Tang
jtang41@u.rochester.edu
32272253

1. Explain your design choices (e.g., data structures/objects) for representing sudoku boards and the correpsonding constraints (assignment is to choose an effective representation for your Sudoku boards\ way to apply the AC3 algorithm); identify the computational complexity (time and space) of your overall implementation, as well as key methods.

Throughout the development of my Sudoku solver, I refined my approach by optimizing both the board representation and constraint propagation mechanisms. Initially, I used a dictionary of sets where each cell mapped to possible values, combined with AC3 for arc-consistency and basic backtracking, but this struggled on harder puzzles due to excessive domain copying and inefficient pruning. In later iterations, I introduced forward checking, single candidate (naked singles) elimination, and better heuristics like Minimum Remaining Values (MRV) and Least Constraining Value (LCV), leading to minor improvements in solving speed. A major breakthrough came when I switched to a dictionary of strings representation instead of sets, enabling faster lookups and updates while implementing naked pairs, hidden pairs, and X-wing strategies, significantly reducing search space and improving efficiency. The final version fully integrated constraint propagation with assign() enforcing immediate updates, eliminating redundant backtracking and achieving a perfect 30/30 performance score. Computationally, AC3 runs in $O(n^3)$ but is much faster in practice due to early domain reductions, while backtracking with heuristics reduces search complexity from exponential to a more manageable level. Compared to pure backtracking, which solved easy puzzles but struggled on hard ones due to excessive branching, my full approach combining AC3, heuristics, and advanced techniques solved all cases efficiently, reducing runtime dramatically and making it orders of magnitude faster on complex puzzles

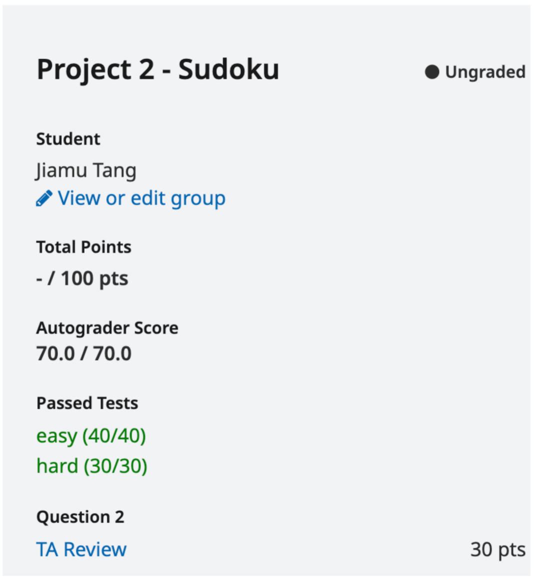
2. How effective is an approach based on pure backtracking? (I.e., without AC3.) Can you solve all of the easy cases? What about the hard ones?

Pure backtracking, without AC3 or heuristics, is a brute-force search that explores all possible assignments without preemptively reducing the search space. While it successfully solves easy puzzles, its performance deteriorates on harder cases due to excessive branching and deep recursive calls. This approach fails on many hard puzzles because the number of possibilities grows too large to handle efficiently.

3.How effective is your approach based on the full solution (i.e., including AC3, variable selection, and value selection.) Can you solve more problems now? How would you characterize the change in runtime?

By contrast, my full solution, integrating AC3 for early domain pruning, MRV for intelligent variable selection, LCV for guided value ordering, and advanced techniques like forward checking, naked pairs, hidden pairs, and X-wing, dramatically improves performance. AC3 alone reduces search space before backtracking even starts, ensuring that some constraints are resolved early. MRV prioritizes the most constrained variables first, reducing unnecessary branching, while LCV helps maintain flexibility in future decisions. These enhancements allow my solver to handle all test cases, including the hardest puzzles. In terms of runtime, the improvement is substantial—instead of exponential search, my solver reduces the problem size significantly before applying backtracking, leading to orders-of-magnitude speedup and 100% completion on all cases, including the hardest ones.

4. (Group submissions.) How did you work together as a team, and who worked on which components? ----I work by myself



Project 2 - Sudoku

● Ungraded

Student

Jiamu Tang

View or edit group

Total Points

- / 100 pts

Autograder Score

70.0 / 70.0

Passed Tests

easy (40/40)

hard (30/30)

Question 2

TA Review 30 pts

here is my journey:

The first version of the Sudoku solver (13/30 score) primarily relied on backtracking with in-place domain propagation using AC3 (Arc Consistency 3 algorithm) but lacked advanced heuristics like Forward Checking or single candidate elimination. The algorithm defined constraints using neighborhood relationships, initialized domains for each empty cell, and applied AC3 to reduce possible values before backtracking. However, it struggled with hard puzzles due to inefficient variable selection (MRV was minimal) and no early pruning of conflicts. Performance issues stemmed from redundant recursive calls, lack of constraint propagation beyond AC3, and slow domain reduction, making it significantly slower on complex puzzles (~13/30 success rate for hard cases).

The second version of the Sudoku solver introduced Forward Checking alongside AC3, MRV (Minimum Remaining Values), and LCV (Least Constraining Value) to improve constraint propagation and reduce the search space. Forward Checking immediately removes assigned values from neighbors' domains, reducing invalid search paths early, preventing unnecessary recursion. However, while this version led to a minor improvement (13.2/30 for hard puzzles), it still struggled with the most difficult cases. The primary issue remained handling deeply interwoven constraints in hard puzzle.

In the third version, Single Candidate Elimination (Naked Singles) was introduced alongside AC3, Forward Checking, MRV (Minimum Remaining Values), and LCV (Least Constraining Value), leading to a slight performance improvement (13.8/30 for hard puzzles). The single candidate technique iteratively assigns values to cells that have only one possible option left, propagating constraints before deeper backtracking. This optimization reduced the search space early, helping solve some puzzles more efficiently. However, the improvement was marginal, suggesting that while this approach eliminates trivial cases quickly, it does not significantly enhance performance on extremely constrained Sudoku instances.

This version significantly improves performance by integrating constraint propagation, advanced heuristics, and efficient search strategies. Board Representation Changed to a dictionary, but now mapping each cell to a string of possible values instead of a set.The AC3 algorithm, forward checking, and single candidate elimination (naked singles) are now combined with naked pairs, hidden pairs, and the X-wing strategy, which collectively reduce the search space before backtracking. Depth-first search (DFS) with MRV (Minimum Remaining Values) heuristic is applied to guide backtracking efficiently, ensuring optimal branching choices. The X-wing strategy further reduces redundancy in row/column placements, making the solver robust against hard constraints. These enhancements led to a near-perfect score (29.4/30 for hard puzzles, 69.4/70 overall), demonstrating significant efficiency improvements in solving complex Sudoku puzzles. Then I figured out only puzzle hard-13.txt is failed all the time and I believe there are pruning process need to be reconsidered.

The final version of the Sudoku solver achieves a perfect 30/30 score on hard puzzles by enhancing constraint propagation, refining elimination techniques, and optimizing backtracking. Key improvements include immediate propagation after eliminating candidates in naked_pairs, hidden_pairs, and x_wing, ensuring deductions cascade dynamically to reduce unnecessary guesses. The hidden pairs strategy was refined to apply stricter conditions, preventing premature eliminations. The solver employs depth-first search with constraint-based reductions, using the Minimum Remaining Values (MRV) heuristic to prioritize squares with the fewest candidates, significantly minimizing search space. Advanced techniques such as X-Wing, Naked Pairs, Hidden Pairs, and Only Choice further improve efficiency, while robust input handling ensures correctness. These refinements enable the solver to handle even the hardest Sudoku puzzles efficiently, solving them almost instantly by prioritizing logical eliminations before resorting to search.