Artificial Intelligence CS 531: Programming Assignment 3

Roli Khanna and Vivswan Shitole

Oregon State University

Abstract. This is the documentation of the third programming assignment in CS 531 Artificial Intelligence (Winter 2019) class. We introduce the problem, and align it with the paradigm of Constraint Satisfaction problems. We then further explain our approach in solving Sudoku with specific inference rules and backtracking through them, as is typically done in Constraint Satisfaction problems. Finally, we present the results of performing forward checking and backtracking over a series of Sudoku problems with varying levels of difficulty (Easy, Medium, Hard), and conclude with answering questions as mentioned in the assignment.

Keywords: Artificial Intelligence · Sudoku · Constraint Satisfaction Problem · Backtracking

1 Introduction

SuDoKu is a constraint satisfaction problem, with all-diff constraints on each row, column, and box. The variables are cells and the values are numbers 1-9. You are going to implement backtracking search with constraint propagation. It maintains all possible candidate values, i.e., the current domain, for each cell after each assignment. The baseline system implements forward checking which removes candidate values for a variable when it violate some constraints, given the currently assigned values for other variables.

The search begins by storing all possible values for each of the empty spots. Then it does constraint propagation through domain-specific inference rules. When the constraint propagation converges, then:

- if no candidates left for some cell, then backtrack from the current state.
- else pick a cell, and assign it a consistent value (keep other choices of values as options to backtrack to; there is no need to backtrack over the cell choice since all cells need to be filled).

We experiment with two different ways of picking a cell.

- Fixed Baseline. Use a fixed order, say, row-wise and top to bottom.
- Most Constrained Variable: Pick a slot that has the least number of values in its domain.

We further apply the following inference rules in the given priority order, i.e., keep applying rule 1 as long as it applies. When it does not, apply rule 2, and go back to rule 1, and so on. The constraint propagation terminates when no rule is applicable, at which point a search action (assignment) is taken. At any search state, the program maintains the set of assignments made in that state, and the candidate numbers available in all other cells. Here are the inference rules to be tried in that sequence.

- Naked Singles
- Hidden Singles
- Naked Pairs
- Hidden Pairs
- Naked Triples
- Hidden Triples

2 Constraint Satisfaction Problems

2.1 Constrain Propagation using Forward Checking

Forward Checking is a constrain propagation scheme in which we maintain the possible candidates in the domain of each variable involved in the CSP. These domains are checked and updated at each variable-value assignment when solving the CSP as a search problem.

For the Sudoku problems, forward checking is implemented by maintaining a "current domain" for each cell enlisting all possible numbers that can be inserted in the cell at the current board state without conflicting with any of the ALLDIFF constraints of Sudoku. With every new assignment of a value to a cell, the domains of all of the affected cells in the same row, same column and the same block are pruned in keeping with the problem constraints.

2.2 Constrain Propagation using K-Consistency

K-Consistency can be thought of as a "multi-step" forward checking scheme for constrain propagation. In this scheme, in addition to pruning the domains of the set of directly affected variables, we recursively prune the domains of the indirectly affected variables due to the primary pruning. Thus this scheme is much faster in propagating the constraints accross all affected variables in a CSP due to a variable-value assignment, as compared to forward checking.

We extended K-consistency on Sudoku by recursively implementing forward checking on all the affected cells due to a particular assignment. This was possible since the consistencies between a cell and all its affected cells in Sudoku is well-defined and modular in nature.

Moreover, advanced inference rules such as Naked Singles, Hidden Singles, Naked Pairs, Hidden Pairs, Naked Triples and Hidden Triples enforce K-consistency in Sudoku. We employed these refined rule-based inference techniques to speed-up the constrain propagation. Hence we observe a big drop in the solving time of Sudoku problems on employing these constraints (evident by the plots shown in the results section).

2.3 Solving CSP using Search and Backtracking

Solving the CSP by formulating it as a state-space search problem is similar to doing a depth first search in a state space where each state corresponds to a particular value assignment to the variables in the CSP. The action space is to select a variable and assign it a legal value, resulting in the next state. Variable selection can be carried out naively (pick the next unassigned variable you encounter) or by using heuristics such as Most Constrained Variable (pick the unassinged variable with the least number of valid candidate values) and Highest Degree Variable (the unassigned variable with most number of arcs for arc consistency with other variables). The value to be assigned to the variable is mostly selected to be the least restricting value. Hence the branching factor is determined by the number of values to be considered for assignment to a particular variable, while the depth of the search tree is determioned by the number of variables in the CSP.

The possible next states obtained can then be explored further with the hope that one of them reach a solution to the CSP. But it is better to check a next state for consistencies using constrain propagation before setting out to explore it further. In case a next state is found to be inconsistent, we can backtrack to the parent state by undoing the variable-value assignment which led to that state. Thus, backtracking using constrain propagation and inference helps to cutoff unnecessarily searches in the sub-trees that do not lead to a valid solution to a CSP. This saves a lot of computation and helps to reach the solution for the CSP faster. Finally, once we have reached a leaf node in the search tree (one with all the variables assigned legal values in keeping with the constraints), we can be sure that we have reached the goal state (solution to CSP).

2.4 Pseudocode for Solving Sudoku as a CSP

The pseudocode for solving Sudoku as a CSP using search and backtracking is given below:

Algorithm 1 Pseudocode for Solving Sudoku as CSP

Input: Initial Sudoku Board
Pick methods: Baseline, MRV
Inference methods: Singles, Doubles, Triples

```
procedure BacktrackSearch(board, depth, pickMethod, inferenceMethod)
    if allFilled(board) then
        if goalState(board) then
            return (True, board)
    if depth == MAXDEPTH then
        return (False, board)
    var = Pick variable using pickMethod
    for val in var.domain do
        newBoard = copy(board)
        insertValue(newBoard, val)
        if propagateConstraints(newBoard, inferenceMethod) then
            flag, newBoard = BACKTRACKSEARCH(newBoard, depth + 1, pickMethod, inferenceMethod)
            if flag then
                return (True, newBoard)
    return (False, board)
```

3 Inference Rules

3.1 Naked Singles

These are cells that have only one possible candidate.Every naked single allows us to safely eliminate that number from all other cells in the row, column, and region that the naked single lies in. The reasoning is that if there is one cell that contains a single candidate, then that candidate is the solution for that cell.

3.2 Hidden Singles

These are the cells which have multiple candidate values in their domain but only one of them will be valid if the constraints are propagated. The extra candidates in the cell "hide" the single solution.

3.3 Naked Pairs

These are a pair of cells in a particular row, column, or region with two identical candidates in their current domain. Hence, the two candidate values can be removed from the domain of all other cells in the same row, column and region as they can only be attributed among the pair of cells.

3.4 Hidden Pairs

These are a pair of cells in the same row, column or region with multiple candidate values in their domain with the property that a unique pair of candidate values occur only in these two cells for the concerned row, column or region. They are "hidden" because the other candidate values in the two cells make their presence harder to spot.

3.5 Naked Triples

These are a group of three cells in the same row, column or region which have only three possible candidate values distributed among their current domains. In other words, the union of their domains is a set of three candidates. Thus these candidates can be removed from the domains of all other cells in the same row, column and region.

3.6 Hidden Triples

These are a group of three cells in the same row, column or region with multiple candidate values in their domain with the property that a unique triplet of candidate values occur only among these three cells for the concerned row, column or region. They are "hidden" because the other candidate values in the three cells make their presence harder to spot.

4 Experiments and Inferences

4.1 Problem setup

Solving a Sudoku is first formulated as a CSP with the cells as variables with fixed domains and the rules of the game as ALLDIFF constraints. Solving the formulated CSP is then in-turn formulated as a state-space search problem coupled with backtracking using inference.

We begin with the initial state as the Sudoku board with the initial entries in place, as defined by the problem instance. The action space is to pick a variable (an empty cell) and assign it a value from its current domain. We follow two heuristics to select an empty cell - pick the first empty cell going in sequential order; or pick the empty cell with the least number of possible candidates remaining in its current domain. The value to be assigned to the variable is selected as the least value in its current domain of valid candidates. This gives us a set of possible next states. Each of these next states are first checked for consistency using constraints propagation and inference rules. This checking is performed recursively over all affected variables for faster constraint propagation. If a next-state is found to be inconsistent, we backtrack to the parent state and repeat the same on another possible next state. Thus we only explore valid next states. This guarantees us that a leaf node (one with all the variables having an assigned value) will be a valid goal state.

4.2 Experiments

We experiment with two different ways of picking a cell.

- Fixed Baseline. Use a fixed order, say, row-wise and top to bottom.
- Most Constrained Variable: Pick a slot that has the least number of values in its domain.

On top of that, we experiment with the following inference rules:

- Naked Singles
- Hidden Singles
- Naked Pairs
- Hidden Pairs
- Naked Triples
- Hidden Triples

4.3 Plots and Inference

All plots are shown with problem difficulty levels along the x-axis and the y-axis showing the output in question. Each curve represents the result for a particular combination of picking a cell (fixed baseline or most constrained variable) with one of the inference rules listed.

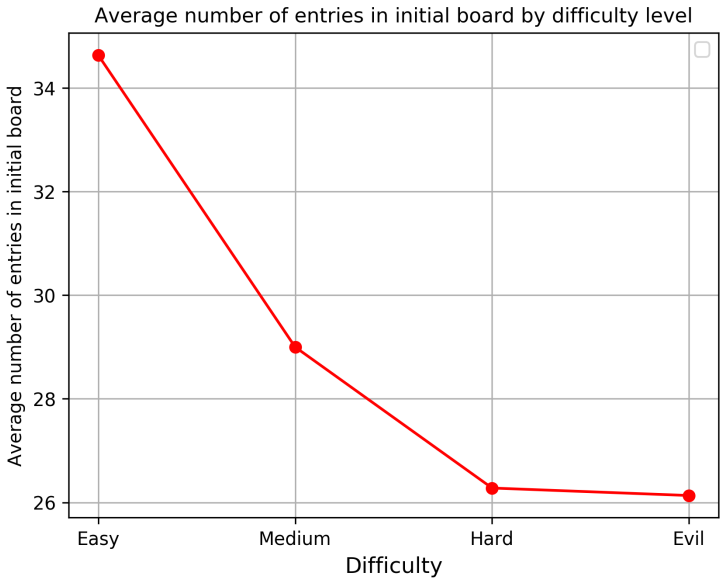

Fig. 1: Average number of entries in initial board - ordered by difficulty levels

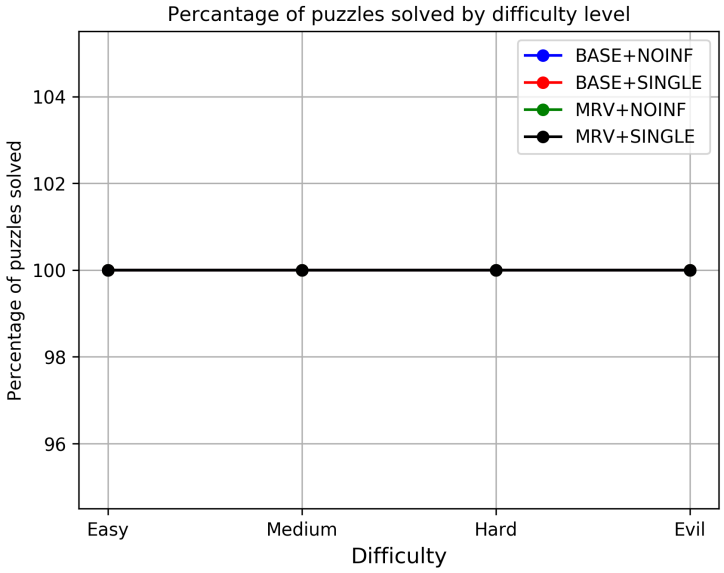

Fig. 2: Percentage of Sudoku puzzles solved successfully - ordered by difficulty levels

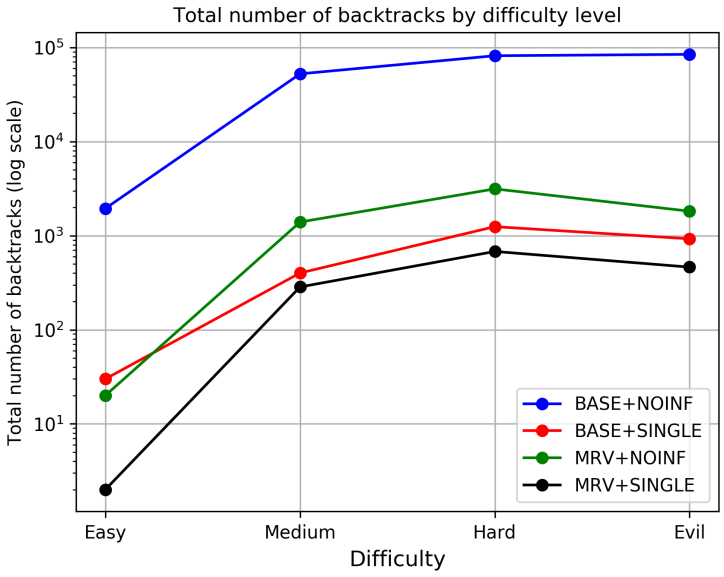

Fig. 3: Total number of backtracks - ordered by difficulty levels

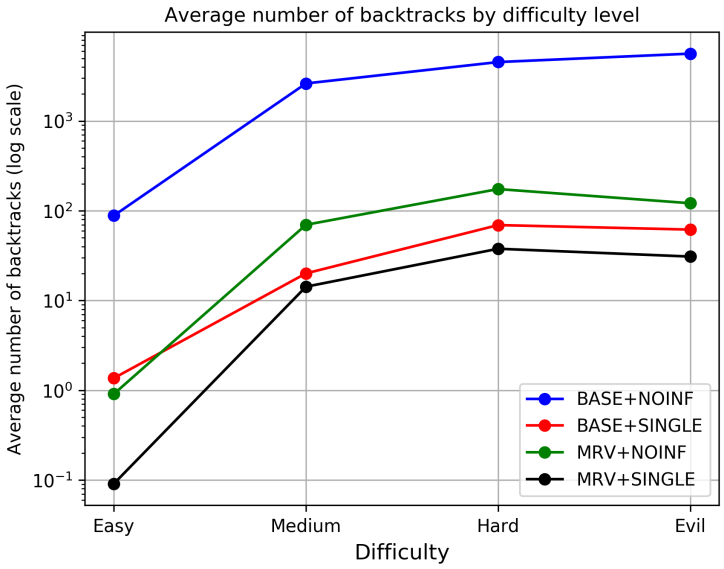

Fig. 4: Average number of backtracks per problem - ordered by difficulty levels

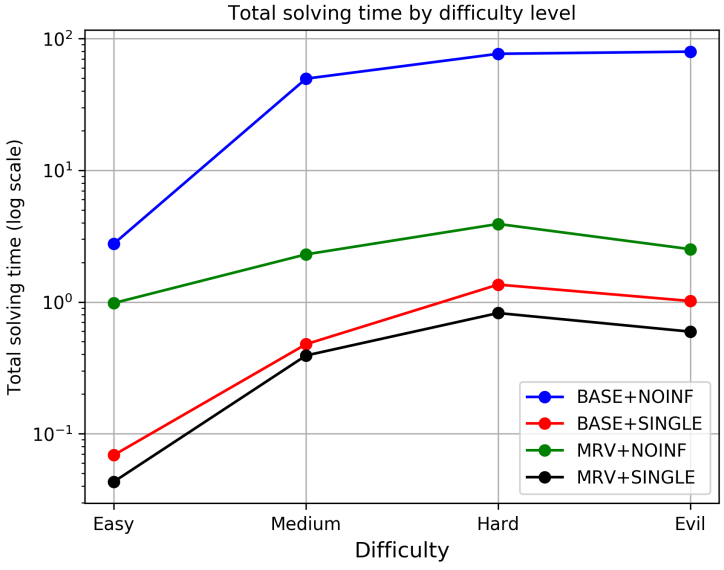

Fig. 5: Total solving time - ordered by difficulty levels

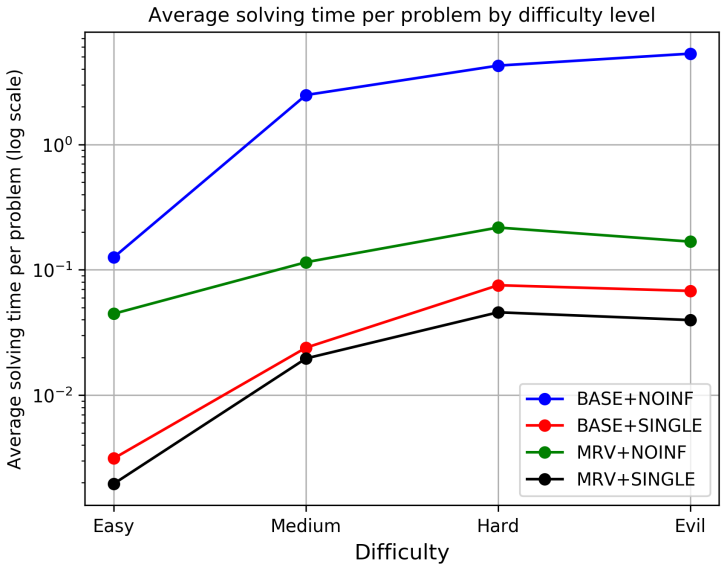

Fig. 6: Average solving time per problem - ordered by difficulty levels

5 Conclusion and Discussion

We formulated Sudoku as a CSP and conducted experiments to solve 77 Sudoku problems of varying difficulty levels using two variable selection schemes (baseline and most constrained variable) and three inference rules (singles, pairs and triples). We obtained the results for the experiments and generated the corresponding plots. Here, we discuss the answers to some of the questions in the problem statement:

1. *Experts appear to grade the problems by the complexity of rules needed to solve them without backtracking. Is this conjecture roughly correct?*

   From the plots comparing total number of backtracks and average number of backtracks with the problem difficulty level, we see that the number of backtracks mostly increase with problem difficulty (except for a slight drop in the number of backtracks in going from hard problems to evil problems). Hence, this conjecture is roughly correct, but not perfectly correct.

2. *Report the average number of filled-in numbers (in the beginning) for each of these types of problems. Would this accurately reflect the difficulty of the problem?*

   The average number of filled-in numbers in the initial board are reported in Figure 1. We see that the curve is monotonically decreasing with problem

difficulty. Hence, the number of filled-in numbers in the initial board accurately reflect the difficulty of a problem.

3. *Discuss how the results vary with the difficulty of the problems*

   As is evident from all the plots comparing a performance metric of the solving algorithm with problem difficulty levels, the general trend is that both the number of backtracks and the solving time increase with problem difficulty. However, each of the search combination is able to solve 100 percent of the problems when using a MAX DEPTH = 1000.

4. *Discuss the effectiveness of the most-constrained variable heuristic compared to fixed selection*

   When using the same inference scheme, a greater number of backtracks and a longer solving time is required for baseline selection (BASE) compared to most constrained variable heuristic (MRV). However, this ordering is not valid if we compare them when each of them uses a inference scheme (e.g. BASE+SINGLE versus MRV+NOINF).

5. *Report on the effectiveness of rule subsets in reducing the search. Is the number of backtracks reduced by increased inference rules? What about the total time for solving the puzzles?*

   The number of backtracks and the total solving time are inversely proportional to the complexity of the inference rules of the problem space. We observe from our experiments that the amount of time taken to solve with no inferences is the highest, and it steadily decreases as we introduce constraints (singles, pairs and triples).