**Group Members: Wah Zhee Dean Justus (A0156231N), Chan Tze Yang Aloysius (A0134202B)**

**CS3243 Introduction to Artificial Intelligence: Assignment 2**

Modelling the Constraint Satisfaction Problem

The problem was modelled into a Constraint Satisfaction Problem in the following manner: all 81 cells are labelled with x-y coordinates, where x and y can take up values in the range {0, 1, 2, 3, 4, 5, 6, 7, 8}, such that (0,0) represents the top left cell, (8,8) represents the bottom right cell, and so on. Additionally, each cell has a domain of digits in the range {1, 2, 3, 4, 5, 6, 7, 8, 9}.

From this point on, there are 3 main constraints that apply to the problem. The first constraint deals with the rows, that no two cells in the same row can take on the same values. For example, the constraint for the top-most row can be represented in the following manner: *AllDiff { (0,0), (0,1) … (0,8) }*. The second row would take on the constraint *AllDiff { (1,0), (1,1) … (1,8) },* and this is repeated for all 9 rows. The second constraint deals with the columns – no two cells in the same column can take on same values. This can be represented in the following manner for the leftmost column: *AllDiff { (0,0), (1,0) … (8,0) }*. Accordingly, this can be repeated to represent the constraint for all 9 columns. The last constraint deals with the smaller 3x3 squares inside the entire grid, no two cells in these sub-squares can take on the same values. For example, the first sub-square can be represented in the following manner*: AllDiff { (0,0), (0,1) (0,2), (1,0), (1,1), (2,1), (0,2), (1,2), (2,2) }*. This constraint applies to all 9 sub-squares.

Explanation of algorithm and heuristic

The algorithm we used combines both the backtracking search as well as the constraint propagation of the arc consistency AC3 algorithm. The entire process of the algorithm is as follows:

1. Initialize the CSP as a graph by creating the variables (all 81 squares in the grid), creating a domain of 1-9 for each variable, and representing the constraints as edges in a graph that connect the domains of the variables together.
2. We read the input sudoku puzzle and update our graph accordingly. While reading the input puzzle, if we chance upon a cell that contains a value other than 0, we reduce the domain of that cell in our CSP to only reflect the single value in the given input.
3. Once we have certain cells with domains of size 1, we run AC3 once on the graph to reduce domains of other cells, via the edges that are affected based on the new domain.
4. When we cannot reduce domains of any other cells further, we start the backtracking algorithm. The order in which the cells are explored in is determined by our *heuristic function*, explained in more detail below.
5. The backtracking selects the first available value based on the reduced cell domain, and temporarily assigns this value to the cell.
6. Once this is done, the AC3 algorithm is run such that all associated cells will have their domains be revised based on this new assignment.
7. The backtracking algorithm then recursively explores other cells in a similar fashion, until it reaches a cell in which has no possible assignment.
8. When this occurs, it has reached a failure case, and backtracks one step, changing the assigned value of the cell to the next available value in the domain.
9. This process continues until we have found a case where every cell is assigned, but there are no failure case and every cell has a possible assignment, meaning that we have arrived at the unique solution for the problem.

The heuristic function that we picked is the Most Constrained Variable, or Minimum-Remaining-Values (MRV) heuristic. Effectively, the backtracking algorithm first selects cells that have the smallest domains, and picks a value in that domain to begin the backtracking search. We picked this because by choosing squares with a smaller number of possible values in the domain, we can effectively reduce the branching factor by increasing our probability of choosing a right value for the cell initially. The logic is as follows: if our cell has a domain size of 2, we have a 1/2 chance of guessing the right value here. This is in contrast to a cell which has a domain size of 3, which we would only have 1/3 the chance of guessing the right value. The significance of “guessing” the right value initially is that we eliminate subsequent branches entirely because we immediately return the solution once we have found a valid one.

For the implementation of our AC3 algorithm, we incorporate forward checking to keep track of the remaining legal values for unassigned variables once we have selected a value to perform the backtrack search on. The forward checking mechanism is what determines when we “fail” and then subsequently “backtrack” before selecting another value.

To decrease the time complexity even further, we formulated a unique approach to backtrack only on the cells with domain size greater than one. Traditionally, the AC3 algorithm would backtrack on every cell, including those that after running the initial AC3, has a domain size of one. By doing this, there is a chance that we can terminate the algorithm (finding the solution) if the domain sizes for the cells are sufficiently reduced to one.

Analysis

The AC3 algorithm has a time complexity of O(n2d3), as determined in the lecture content. As we run the AC3 algorithm for every level of the backtrack search, which has a worst-case exploration of O(nd) time complexity, because every domain for every variable is visited once, we have a worst-case time complexity of O(n3d4). Fortunately, because of the improvements that we made in terms of the heuristic we chose, as well as a given number of cells that are already filled up, and how we run AC3 once initially to reduce domains even further, the average time complexity is significantly smaller. Algorithms that are similarly efficient include the dancing-links algorithm, unfortunately this algorithm does not consider the problem as a CSP and solve it from there. Some of the other heuristics in which we considered in the process of designing this algorithm include the least constraining value, which would see us select the variable with most constraints on the remaining unassigned variable. Unfortunately, this algorithm would be hard to implement as it is hard to determine which cell has the least constraints on the remaining unassigned variables, without incurring significant time complexity cost understanding which of the cells is the most constraining variable. Domains would have to be compared with every other domain that the cell is associated with. Additionally, it would not allow us to “fail early”, suggesting that it is not as efficient as the MRV heuristic that we chose to implement for this CSP. A similar argument can be made for the most constraining variable. Logically, it would make most sense to choose the minimum-remaining-values heuristic as it is easier to implement, and is extremely efficient.