Hansen Tamara Dharmawan | A91413023
Mark Darmadi | A11410141
Vania Christie Chandra | A12496582

CSE 150 Assignment 3 write up

Description of the problem and the algorithms used in the problems (culminating in the solution in Problem 6).

For part 1, we are required to implement the checker to determine whether all the variables have been assigned and hence completed. The implementation for this function was quite straight forward, the hard part was reading through the API and figuring out what functions we needed. Once we figured that out, all we had to do was iterate through all the variables and checking if each variable has been assigned or not. As soon as we find one variable that is unassigned, the function will immediately return false.

For part 2, we are required to check if the variable assignment to the value is consistent. Similar to part 1, we iterate through all the variables and we first check if that variable has been assigned or not, only then do we check using the given function is_satisfied to see if the assignment is consistent with the value. Once everything has been satisfied, we confirm that the variables have been assigned consistently.

For part 3, We are implementing the backtrack search algorithm. This algorithm takes the csp problem and search for a solution for the csp. First, the algorithm check if all the variable has been assigned using the is_completed function. Then the algorithm select one of the unassigned variables and try to assign it values from its domain. Note that in part 3, the method of selection is still very simple (not implemented yet). On each value, the algorithm assigns it to the variable first, and then find the inferences that results from this assignment. If all the inferences are valid, the algorithm will then recursively calls backtrack function. If the returned value is true, the algorithm returns True, indicating that a solution has been found. If returned value from the recursive call is False, the algorithm will then tries another value for the chosen variable, and repeat the process.

For number 4, we are implementing the arc consistency algorithm (or ac3). The algorithm that we followed can be found in the slides that discuss the constraints satisfaction problems. First, the input of the algorithm is the CSP itself and the arc to be maintained. If arc is not given, we can take the arc of the constraints in the CSP. For all the arcs, we check one by one if the pair of variable need to be revised (there is no value that can satisfy the constraints). If so, we delete that particular domain from the list of domains Xi can be in. If a particular arc needs to be revised, we then get the list of neighbors and add it to the queue so they can be further check to see if there is further revision.

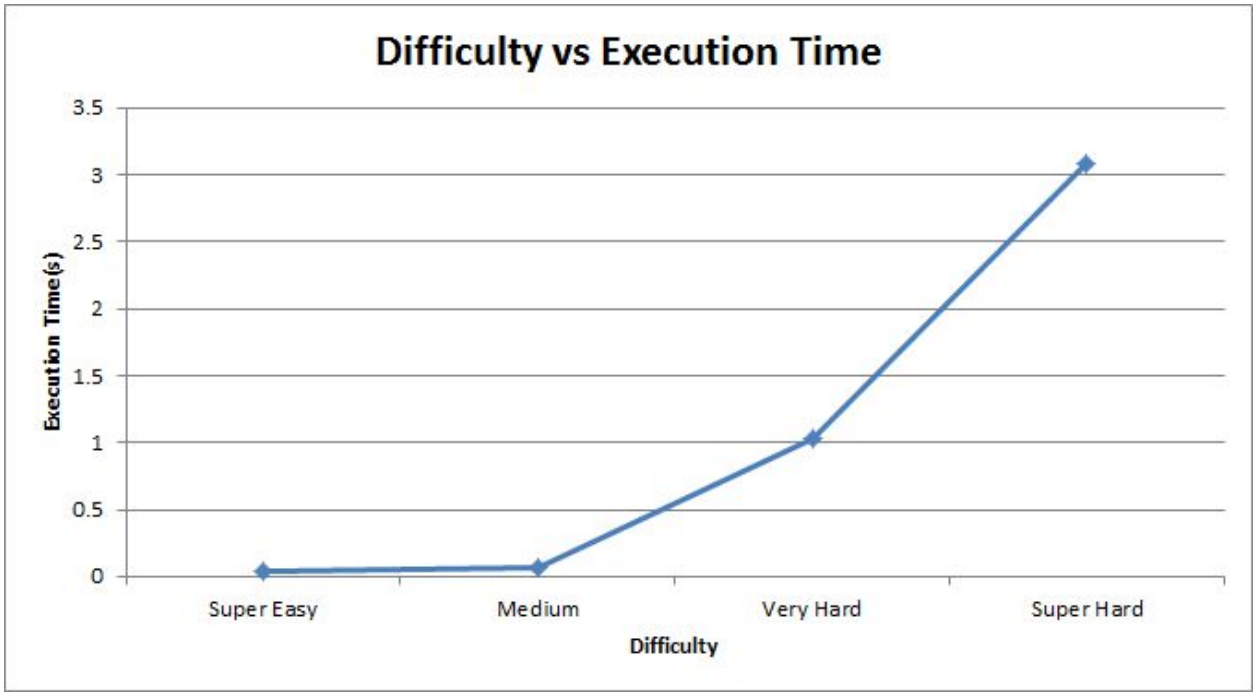
For part 5, we have to implement the variable and value ordering heuristics. We will use the minimum-remaining-values (MRV) and the degree heuristic as a tie breaker. Firstly, I will go through all the variables to find the first unassigned variable, for every unassigned variable, i push that variable into a list which I will later manipulate. If all the variables are assigned, we can then return None because there is nothing to assign. If there is at least one variable to assign, we then sort the list, with the lowest number of domain remaining. With that, we want to check if the first element of the list have other variables that have the same domain count. So we go through the list and check if the domain size is the same as the first element in the list. If there are no variables with the same domain size, then we can say that there is not a tie and we can return the head of the list. If there is a tie, we then have to go through the tie list and check each variable with the highest constraint and we return that variable. For the value ordering, we use the least-constraining-value (LCV) heuristic to solve this. I first create a dictionary and fill it with the domain element in the given variable's domain, I initialize the count of all the elements to 0. From there we can get the list of constraints involving that variable and then we count the number of times each domain element occurs in each of the constraints, we then store it in the dictionary each round of iteration. After the dictionary is filled with all the values, we can sort the dictionary according to the lowest count and we can push them one by one into a list and return that list afterwards.

For part 6, We implemented the improved backtracking search by combining all the implemented functions from part 1 to 5. The completed algorithm takes into account the arc consistency and the order of chosen variable as well as the chosen value for that variable. By optimizing those aspects, the algorithm should run faster and more efficiently.

Run the solver you developed on different types of puzzles and measure how the solution time varies with the board size and difficulty ratings. Show your results in two separate graphs (one for the board size and another for the difficulty rating). Summarize your finding in a paragraph.

Size	Time (s)
4x4	0.004000
6x6	0.042000
8x8	7.513999
9x9	14.2359

Difficulty	Time (s)
Super Easy	0.0429999
Medium	0.0629999
Very Hard	1.0369999
Super Hard	3.0869999



As expected, as board size gets bigger, the execution time of the program also gets longer. The relation between the board size and the execution time seems to be exponential because by going from 6x6 to 8x8, the execution time increases dramatically from 0.042s to 7.51s. Similarly, as difficulty rating of sudoku gets higher, the execution time of the program gets longer. The relation between the difficulty rating and the execution time seems to be polynomial because the execution time does experience a drastic increase albeit not as dramatic as the previous relation.

Analyze the effect of each type of heuristics (inference, variable and value ordering). Compared to the basic solver in p3, how much does each type of heuristic speed up the solver for this problem?

All 3 type of heuristics should increase the speed of the program. The inference heuristic speed up the program from basic algorithm (part 3) on 3.779 s to 2.68 s. This is a considerable increase that will be useful on bigger data scale. The variable heuristic suppose to speed up the algorithm. However, our implementation of the variable heuristic is imperfect that it does not actually improve the speed of the program. The value ordering heuristic speeds up the program by a small amount. The program is sped up from 3.7539 s in the basic algorithm to 3.6210 s if only value ordering is implemented.

A paragraph from each author stating what their contribution was and what they learned.

Hansen Dharmawan - I was in charge of debugging and implementing part 3 and 6. I learn how the whole algorithm works and how everything comes together to solve the CSP faster.

Mark Darmadi - I was in charge for implementing part 1, 2 and 5. I learn how to use the API for this project. And I also familiarized myself with the various ways of sorting list and dictionary.

Vania Chandra - I was in charge of implementing part 4 and 6. I learn how to implement the AC3 algorithm as well as the overall flow of backtracking algorithm.