Yoon Ho Han, Kevin Zhang
yhhan@uucsd.edu, ktzhang@ucsd.edu

Problem 1:

Implement the is_complete method that returns true if all variables have been assigned. In order to check if all the variables of the CSP were assigned, the algorithm we wrote iterates through every variable in the CSP and checks if that variable is_assigned(). If any aren't assigned, return false. If all variables are assigned, the algorithm eventually exits the loop and returns true.

Problem 2:

Implement the is_consistent method that returns true if the variable assignment is consistent with all the constraints on it. We wrote an algorithm that loops through all the constraints on the given variable. For each constraint, we loop through all the variables to find a matching variable for the constraint. If the assigned value of the new variable and the value passed in for the given variable don't satisfy the constraint, return false. Otherwise, the algorithm has checked all constraints for the given variable and has found no inconsistency, so return true.

Problem 3:

Implement a basic backtracking algorithm. The algorithm we wrote to do this is a recursive function that calls itself after assigning a value to a variable. In the backtracking algorithm, if the board is complete, return true. If not, then select the next unassigned variable and check all the domain values for that variable. If a value is consistent, begin_transaction() so that if this value doesn't work later on, we can go back to this board state. After assigning the value to the variable, call backtrack on the new board. If the call to backtrack succeeded, return true, if not, rollback to the most recent begin_transaction(). If the whole algorithm runs without finding a solution (returning true), then there is no solution, return false.

Problem 4:

Implement the ac3 algorithm. In the ac3 algorithm, we have a queue of all the arcs in the csp or the given arcs if it was passed in. Iterating through the arcs in the queue, check if the arc needs to be revised. We call a separate revise() function that takes in 2 variables and returns true if the domain was revised. In the revise function, check all values for xi's domain. If no value of xj's domain satisfies the constraint, then delete that value from xi's domain and set revised to true. In the ac3 function, if the domain was revised, check if domain is 0; if it is, then return false, arc consistency check failed. If it did not fail, iterate through all of xi's neighbors and add the arcs between xi and its neighbor to the queue.

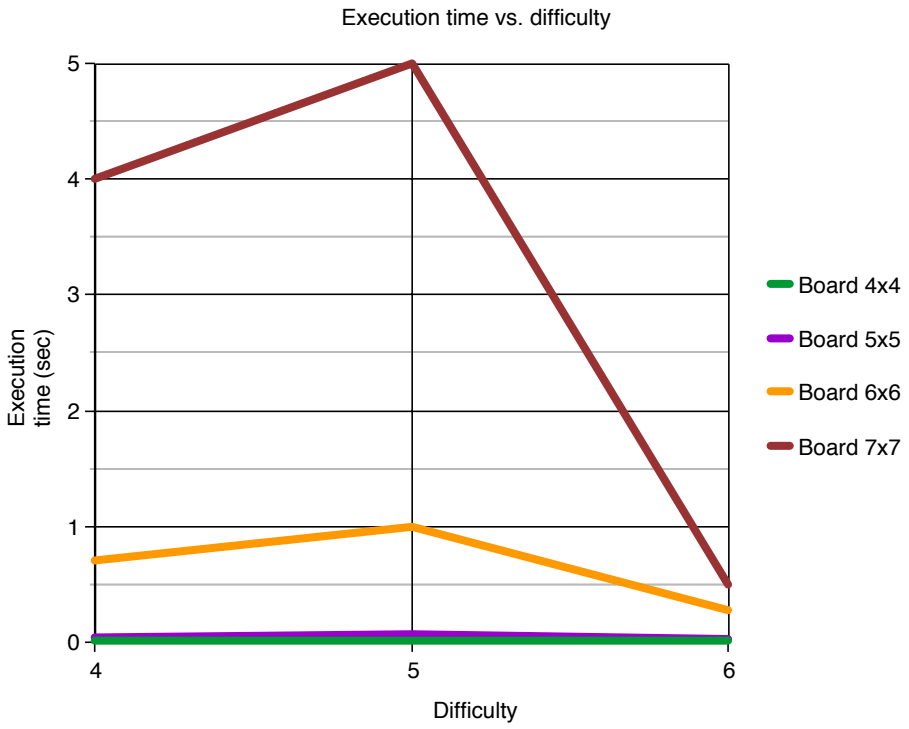
Problem 5:

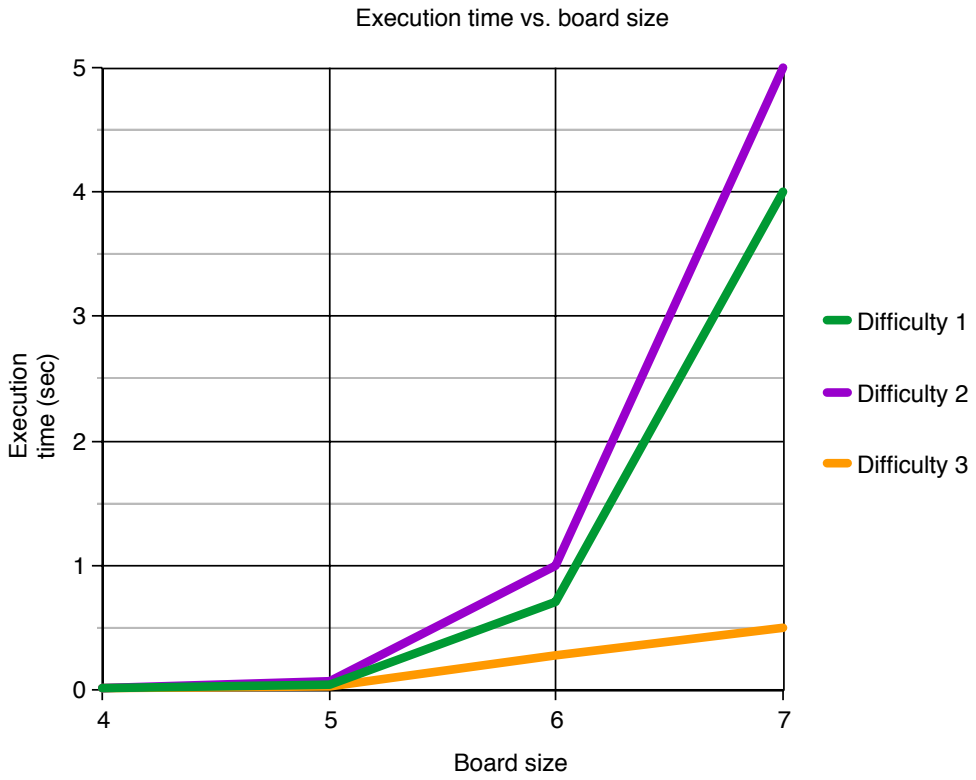
Implement the variable and value ordering heuristics. The variable ordering is based of the minimum-remaining-values heuristic; if there is a tie, choose the variable involved in the most number of constraints. To select the next unassigned variable with MRV, keep track of the variables with the current smallest domain in a

queue. For every variable in the csp that's not assigned, if the length of the domain is less than the current smallest domain, then clear the queue, update the smallest domain to current length of domain, and append current variable to queue. If the variable has equal size domains and the smallest domain, then append it to the queue. If the variable has a larger domain than smallest domain, it is simply skipped. After iterating through all the variables, the queue holds all variables with equal, smallest domains. Among them, choose the variable that has the largest size of constraints.
The value ordering is based on the least-constraining-value heuristic. Loop through possible values for the given variable. For each value, check all neighbors of the variable to count the number of conflicts the value would create (this is done by checking if neighbors have the same value in its domain)(also, the value for the given variable is counted to simplify the code and not have to check if the 1st variable or 2nd variable of the arc is the neighbor). With the value and the number of conflicts, append a tuple to a list holding value-to-conflicts. After iterating through all values, sort the list in ascending conflicts. Then, loop through list and create a new domain of values, in order of increasing conflict.

Problem 6:

Complete a faster backtracking search algorithm by augmenting the basic backtracking algorithm with the MAC inference and the variable and value ordering heuristics written so far. To do this, use variable and value ordering heuristic of part5 to choose the next unassigned variable and order the values of the domain to try assign. Also, after every assignment, call the inference method that runs ac3 and deletes values of domains that are impossible.





Based on these results, increasing the board size consistently increases execution time. This is reasonable, because the algorithm needs to search through more possibilities to solve the puzzle. The most interesting thing about the data is that difficulty and execution time was not linearly dependent. We assumed that as difficulty goes up, the execution time would go up as well. However, after a certain point, raising the difficulty actually sped up the execution. This is probably because of the inference method that is able to eliminate much more possible values for each variable, thus decreasing the possible states.

Yoon Ho Han:

I did the typing for the logic of most of the algorithms. While we worked on the parts together, I contributed most to the ac3 algorithm and variable/value ordering heuristic. Through this assignment, I learned that copying pseudocode to actual code is harder than expected. I realized that I need to actually understand how the algorithm works before implementing it. This is because I ran into problems later on for not understanding exactly how the algorithms ran.

Kevin Zhang:

I set up the necessary components to work on the assignment together, like GitHub. I helped with a lot of the logic of the algorithms, and my partner did a lot of the typing this time. I learned more about the uses of arc consistency and backtracking algorithms through this assignment. In addition to knowledge about these search algorithms, I gained more experience in Python syntax.