CS171 Project Progress report #3: Arc Consistency

Name: Haoming Huang, Jason Chen-Ju Hsieh

Date: February 21, 2016

1 Scope

The purpose of this assignment is to implement the Arc Consistency component of Sudoku solver. Our team will be continuing off of the provide Java shell and changes made from the last assignment.

2 Progress

We've deleted LocalSolver.java and Assignment.java along with the pushAssignmnet() function from ConstraintNetwork because these are for extra credit which we will not be doing, except for Arc Consistency. We've also removed a small error in calculating the preprocessing time.

We've implemented Arc Consistency based on the pseudocode presented in the lecture slides, using a queue. Again we only check variables that are already assigned since only those can be used to remove domain values of neighbors. While the queue is not empty, we check the neighbors of assigned variables, we remove constraining domain values from neighbors only if they're not assigned. After removing domain values, if the neighbor only has one domain value left (in other words, assigned) and it's not present in the queue already, we add that neighbor to the queue to be checked later. If the neighbor has the same assigned value as the original variable, then return false and backtrack.

3 Problems & Questions

At first the program kept getting stuck in infinite loops and it was very difficult to figure out why. We thought that we could fix it by resetting the modified status of variables to false after deleting domain values, but that introduced a new problem where it mostly works but an already modified variable kept getting added back into the queue. Later we found that the solution is actually inconsistent. Eventually we realized that we did not check whether a variable is already in the queue, inconsistent assignments, and whether a variable is assigned or not before deleting domain values, since at first we thought that domain values cannot be deleted if the variable is assigned, and we set it to return false if variable's domain becomes empty. After applying all those fixes, we set it so that an inconsistent assignment would return false and backtrack.

4 Results

The program takes in correct inputs (input file, output file, timeout, and ACP (arc consistency preprocessing, where AC is done only once on the initial puzzle) or MAC (maintain arc consistency, where AC is done every time as new values are assigned), correctly solves the Sudoku puzzle using the back track search implementation and uses arc consistency to remove inconsistent domain values to speed up the searching process, and outputs a file with the correct format. Back track search with MAC is slightly faster with slightly less node counts. However the bigger puzzles still take too long to solve so we expect the differences to be higher there.

Appendix:

```
private boolean arcConsistency()
{
        for(Variable v : network.getVariables())
        {
                if(v.isAssigned())
                {
                        Queue<Variable> acq = new LinkedList<Variable>();
                        acq.add(v);

                        while (!acq.isEmpty())
                        {
                                Variable n = acq.remove();
                                for(Variable vOther : network.getNeighborsOfVariable(n))
                                {
                                        if (!vOther.isAssigned())
                                        {
                                                vOther.removeValueFromDomain(n.getAssignment());
                                                if (vOther.isAssigned() && !acq.contains(vOther))
                                                        acq.add(vOther);
                                        }
                                        if (n.getAssignment() == vOther.getAssignment())
                                                return false;
                                }
                        }
                }
        }
        return true;
}
```