CS171 Project Progress report #5: LCV Heuristic

Date: March 06, 2016

1 Scope

The purpose of this assignment is to implement the LCV 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 implemented LCV based on explanations from lecture. LCV attempts to decide which value to assign to a variable out of the possible values to assign. The decision is based on how much that assignment affects remaining variables. In this case, neighboring variables of the Sudoku Puzzle. LCV uses a comparison function that counts how many neighboring variables each possible assignment value affects and uses that to decide whether one assignment value is better than the other. If an assignment value affects more neighboring variables then it's less favorable, since it's more constraining. This comparison function is then used to sort the list of possible assignment values and return them in an order from the least constraining to most constraining.

3 Problems & Questions

Initially we thought that LCV is choosing which variable to assign values to, rather than choosing which value to assign to a variable, so we wrote code that returns the list of neighboring values order by the size of their domains, we realized our mistake afterwards.

It was still difficult to decide how to implement LCV programmatically. Looping through the assignment values seemed like a bad practice, but it was difficult to figure out how to use a comparator instead. When the function was first tested there was a check to immediately return an assignment value as most constraining if it has an assigned neighbor with the same value as the assignment value, we then realized that a better practice would to ignore neighbors that are already assigned, this would mean less checking, and it would still work when there are no constraint checks being carried out (such as FC).

Our implementation was also wrong as we had an "else if" when there was supposed to be an "if". Strangely this led to less node count using the test file PH5, however testing with PH4 we realized that it was the same as if LCV is not being used. This error was discovered after carefully inspecting the code for any final problems.

4 Results

The program takes in correct inputs (input file, output file, timeout, and LCV (least constraining variable)), correctly solves the Sudoku puzzle using the back track search implementation along with heuristic functions to choose what value to assign to a variable that is most likely to lead to success. When LCV is added to ACP, MAC, MRV, DH, the solving speed is up to 3 times faster and count nodes are up to 3 times less for the most difficult puzzles.

Appendix

```
public List<Integer> getValuesLCVOrder(final Variable v)
{
	List<Integer> values = v.getDomain().getValues();

	Comparator<Integer> valueComparator = new Comparator<Integer>(){

		@Override
		public int compare(Integer x, Integer y) {
			Integer xcount = 0, ycount = 0;

			for(Variable vOther : network.getNeighborsOfVariable(v)) {
				if(!vOther.isAssigned()) {
					for(int each : vOther.Values())
					{
						if(each == x)
							xcount++;
						if(each == y)
							ycount++;
					}
				}
			}

			return xcount.compareTo(ycount);
		}
	};
	Collections.sort(values, valueComparator);
	return values;
}
```