7. Report Template (Required: Turn This In As Your Report)

[Anything in **boldface** below is required; anything below in angle brackets <> is a parameter; and anything below in square brackets [] is an instruction. The required file format for document transmission is always PDF. Students might write the report in Word or some equivalent, but thereafter they **must** convert their final document version to PDF for submission.]

[Turn in only **ONE REPORT PER TEAM** --- not one per person. Please follow the instructions on the class website about naming and file formats. Nonconforming submissions will lose points.]

Preamble: You, and How to Run Your Code

[Use no more than 1 page for this section.]

My name, ID#, UCInetID: Malav Pandya, 91044866, mcpandya@uci.edu
Partner name, ID#, UCInetID (or "none"): None
By turning in this assignment, I/We do affirm that we did not copy any code, text, or data except CS-171 course material provided by the textbook, class website, or Teaching Staff.

The programming language(s) and versions you used in your project:
Java 1.7
The environment needed to compile and run your project:
Eclipse and openlab.ics.uci.edu

A small write up of your implementation.
All the techniques I have implemented (FC, MRV, DH, MRV_DH, LCV, FC_CP) have comments respectively associated with them. Overall, I have mainly used the Java shell provided to carry out my solver. I have also implemented mytest.java, which basically covers the how the tokens are used, computes the result and puts in the output file, etc. Also, I have tweaked some parts of the shell in order to improve the proficiency of my algorithms.

Part 1: (Required) What You or Your Team Did

[Fill in the check box below to indicate what you or your team did. Put exactly one "X" in each triplet of Yes/Partly/No for each line item. Each Yes/Partly/No triplet must have exactly one "X".

Please note: You will lose many points if you claim to have done something the Teaching Staff cannot reproduce. Please check **exactly one** of Yes/Partly/No for **each** item and condition below.

Please note: If you answered "I/We tested it thoroughly" as not "Yes" then you ***must*** answer "It ran reliably and correctly" as "No" because "Not tested thoroughly" ⇒ "Not reliable."
You will lose points if you say it ran reliably and correctly but you did not test it thoroughly.]

I/We coded it.			**I/We tested it thoroughly.**			**It ran reliably and correctly.**			**What was it?**
Yes	Partly	No	Yes	Partly	No	Yes	Partly	No	
Required Coding Project									
Yes			**Yes**			**Yes**			Backtracking Search (BT)
Yes			**Yes**			**Yes**			Forward Checking (FC)
Yes			**Yes**			**Yes**			Minimum Remaining Values (MRV)
Yes			**Yes**			**Yes**			Degree Heuristic (DH)
Yes			**Yes**			**Yes**			Least Constraining Value (LCV)
Extra Credit									
									Writing Your Own Shell
									Writing Your Own Random Problem Generator
									Arc Consistency AC-3/ACP/MAC
									Local Search using Min-Conflicts Heuristic
							Partly		Advanced Techniques, Extra Effort, or Creativity Not Reflected Above (*)

(*) Advanced Techniques, Extra Effort, or Creativity Not Reflected Above:
[If you/your team implemented any advanced technicques, made any extra effort, or exhibited any extra creativity, that is not reflected above, please describe it briefly but clearly here.]

Part 2: N=9 (9x9) Sudoku: Analysis of Best Methods Combination

Fill in the following table based on your system's performance on the "hard" 9x9 problems: PH1, ..., PH5 on the supplied coding shell. For each row in the table, run your system on PH1-5 with the indicated option tokens turned on, then report the average number of nodes expanded, the average time taken, and the standard deviation of the time taken. The blank line at the top is for the case of using backtracking search only; For breaking ties, order the variables lexicographically, and order the values 1 to 9. For the next seven lines, each with only one option token, run your system with only that single token enabled. (ACP, MAP, and Other are optional; if you did not do them, leave them blank.) On the next line, run FC, MRV, DH, and LCV all together. Finally, do experiments with other combinations of methods, and fill in the last blank

line to indicate the combination you found to be fastest by putting an "X" under any option token that it uses.

FC	MRV	DH	LCV	(ACP)	(MAP)	(OTHER)	AVERAGE # NODES	AVERAGE TIME	STD. DEV. TIME
The blank row below is for the case of no heuristics and no constraint propagation.									
							2266344	**120.5s**	**182.95s**
X							**50786**	**2.75s**	**5.58s**
	X						**1774410**	**77.6s**	**163.57s**
		X					**2369370**	**88.3s**	**134.57s**
			X				**1652380**	**150.7s**	**228.67s**
				(X)					
					(X)				
						(X)	**272**	**0.07s**	**0.10s**
X	**X**	**X**	**X**				**19961**	**1.42s**	**3.2s**
X						**(X)**	**272**	**0.07s**	**0.10s**
Fill in the blank row above with the combination you found to be fastest on PH1-5.									

Did you get the results you expected? Why or why not?
Yes, I did get the results that I expected. I knew that forward checking alone would require less explored nodes than backtracking. And since the rest of the techniques were called individual, I expected them to do worse than FC. I was surprised by how slow MRV and DH were individually. Especially MRV, which had more average nodes than simple backtracking.

Did you implement any Advanced Techniques, Extra Effort, or Creativity not reflected above? If so, please tell us what you did.

As an experiment, I added constraint propagation to my forward checking function, which makes the solver a lot faster. Basically, every time FC would be called, constraint propagation would delete domain values from unassigned variables which were inconsistent with the already assigned variables. As a result, it ended up working better than all my other combinations.

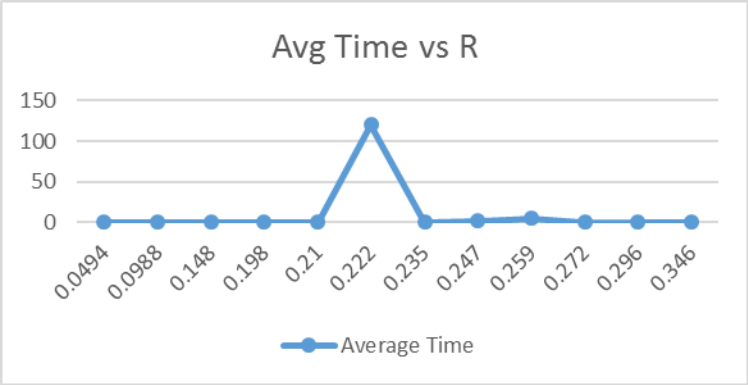
Part 3. N = 9 Sudoku: Estimate the Critical Value "Hardest R"

3.1. Fill in the following table, for total time, where $R = M / N^2$.

[Average together at least 10 different puzzles for each data point. Feel free to use different data points if it yields a more informative picture of your system running its best combination above.]

M	N	P	Q	R = M / N^2	AVERAGE # NODES	AVERAGE TIME	STD. DEV. TIME	# (%) SOLVABLE
4	9	3	3	0.0494	79.7	0.0311	0.026	100
8	9	3	3	0.0988	47.7	0.024	0.020	100
12	9	3	3	0.148	55.2	0.024	0.014	100
16	9	3	3	0.198	36.5	0.026	0.014	100
17	9	3	3	0.210	273	0.065	0.092	100
18	9	3	3	0.222	346677	120.01	183.29	70
19	9	3	3	0.235	33.5	0.023	0.016	100
20	9	3	3	0.247	1911	2.2	1.81	80
21	9	3	3	0.259	17758	5	4.12	50
22	9	3	3	0.272	2991	0.21	0.54	70
24	9	3	3	0.296	670	.02	0.03	50
28	9	3	3	0.346	15.7	0	0.0018	10
32	9	3	3	0.395	10.4	0	0	0
36	9	3	3	0.444	9	0	0	0

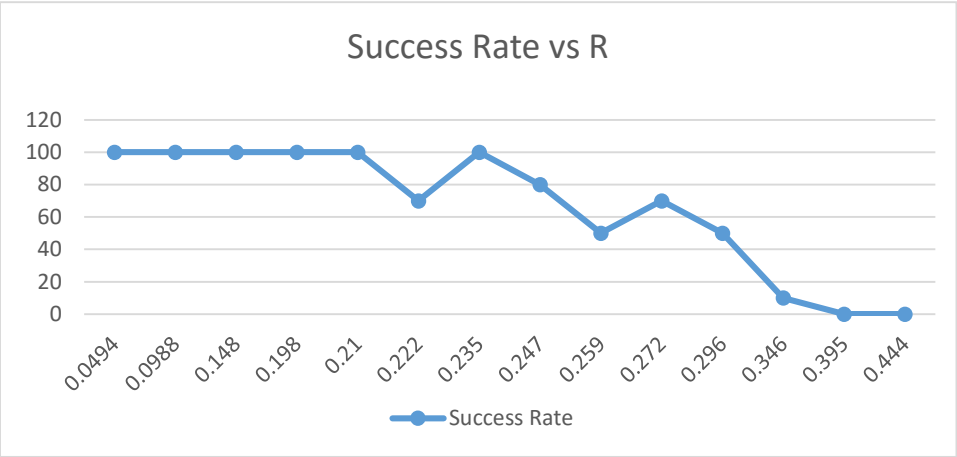
3.2. Find the critical value of the "hardest R" for N = 9 and your best combination above. [Based on your data in 3.1 above, produce a graph similar to that shown below.]



Based upon your results above, you estimate the "hardest R_9" = 0.222

3.3. How does puzzle solvability for your best combination vary with R = M / N^2? [Based on your data in 3.1 above, produce a graph similar to that shown below.]

As R increases, my solvability % decreases. I assume, my puzzle inclines to fail as more assignments are initially assigned to the Sudoku (Large M). If I tested on more examples, I believe I would find a gradually decreasing curve to show the relationship between R and the solvability %.



[For you, the x-axis will be R = M / N^2, and the y-axis will be the probability that a randomly generated **and completed** Sudoku puzzles is solvable using your best combination above.]

3.3 Is the critical value for "hardest R" approximately the same as the value of R for which a random puzzle is solvable with probability 0.5?
Answer: No because the hardest R I found (R = 0.222) had a solvability % of 70. However, the second hardest R (R = .259) did have a solvability % of 50.

4. What is the "Largest N" You Can Complete at the "Hardest R_9"?

Fill in the following table using your best combination of methods from Part 2 above. Fix a time limit of **5 minutes or less** to complete each new puzzle. For each new value of N, scale M with the R_9 you found in Part 3 as:

Scaled M = round ($N^2 \times R_9$)

Consider at least 10 random puzzles for each new value of N. Report the number and percentage of random puzzles that your best combination was able to complete for that N in **5 minutes or less.** For those puzzles that your system completed, report average nodes, average runtime, and standard deviation of the runtime. (Standard deviation is "none" if your system completed only one puzzle.) When your system fails to complete any puzzles for some value of N, it is unnecessary to continue.

"Hardest M" round ($N^2 \times R_9$)	**N**	**P**	**Q**	**# (%) Completed in 5 Minutes or Less**	**AVERAGE # NODES (Completed puzzles only)**	**AVERAGE TIME (Completed puzzles only)**	**STD. DEV. TIME (Completed puzzles only)**
31	12	3	4	20	80493	14.29	10.6

50	15	3	5	0	0	0	0
57	16	4	4	0	0	0	0
72	18	3	6	0	0	0	0
89	20	4	5	0	0	0	0
98	21	3	7	0	0	0	0
128	24	4	6	0	0	0	0
162	27	3	9	0	0	0	0
174	28	4	7	0	0	0	0
200	30	5	6	0	0	0	0
227	32	4	8	0	0	0	0
272	35	5	7	0	0	0	0

Part 5. Monster Sudoku: Is "Hardest R" constant as you scale up?

I was not able to do part 5 the way it was specified because I was not able to complete all puzzles in 5 minutes or less in part 4. However, I did solve 20% of puzzles for N=12 so I used it for part 5.

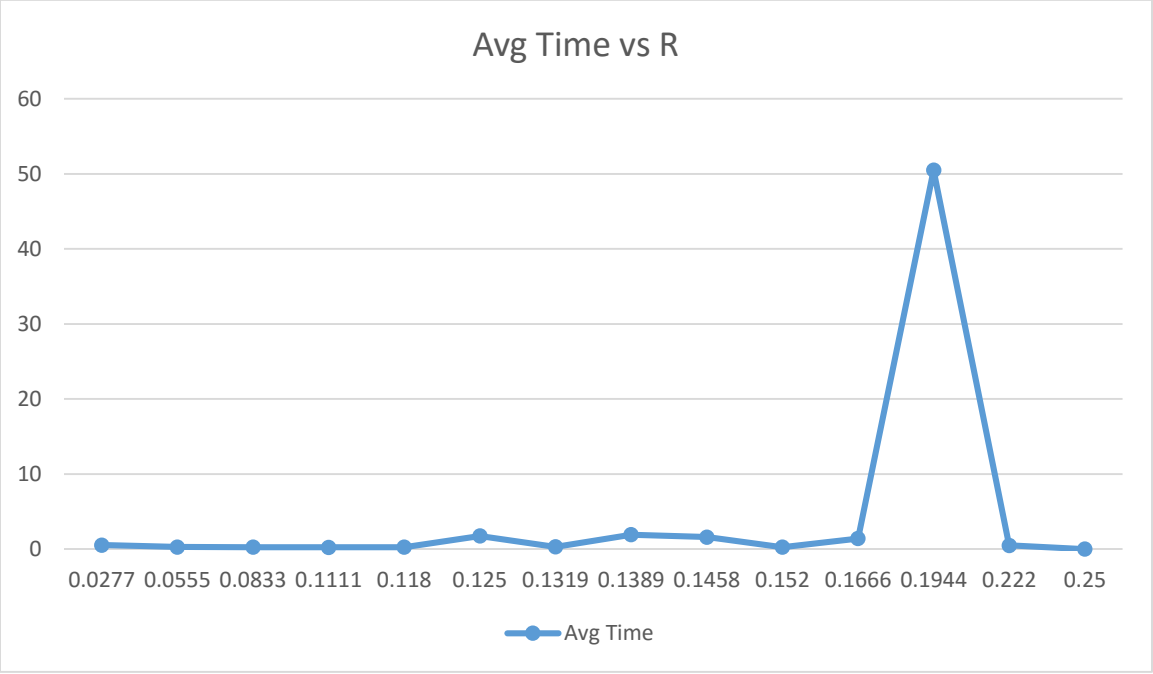
5.1. Fill in the following table.

[Fill in N, P, and Q from the largest value of N in Part 4 for which you **completed all puzzles in 5 minutes or less**. Then, try different M values that are computed from R values tried in Part 3. For example, new M values can be round (0.0494 * N_largest^2), round (0.0988 * N_largest^2), etc. Note that 0.0494, 0.0988, are came from the first two rows in table in Part 3. The actual values of M that you use will result in slightly different values of R because of rounding, so recalculate R below for the values of M you actually used. Average together at least 10 different puzzles for each data point. Feel free to use different data points if it yields a more informative picture of your system running its best combination above.]

M	**N**	**P**	**Q**	**R = M / N^2**	**AVERAGE # NODES**	**AVERAGE TIME**	**STD. DEV. TIME**	**# (%) SOLVABLE**
4	12	3	4	.0277	**14512.1**	**0.53**	**0.53**	**90**
8	12	3	4	.0555	**792.4**	**0.275**	**0.216**	**90**
12	12	3	4	.0833	**895.4**	**0.26**	**0.318**	**90**
16	12	3	4	.1111	**548.3**	**0.239**	**0.212**	**60**
17	12	3	4	.1180	**638.5**	**0.259**	**0.096**	**60**
18	12	3	4	.125	**3136.3**	**1.75**	**4.265**	**80**
19	12	3	4	.1319	**663.7**	**0.303**	**0.47**	**90**

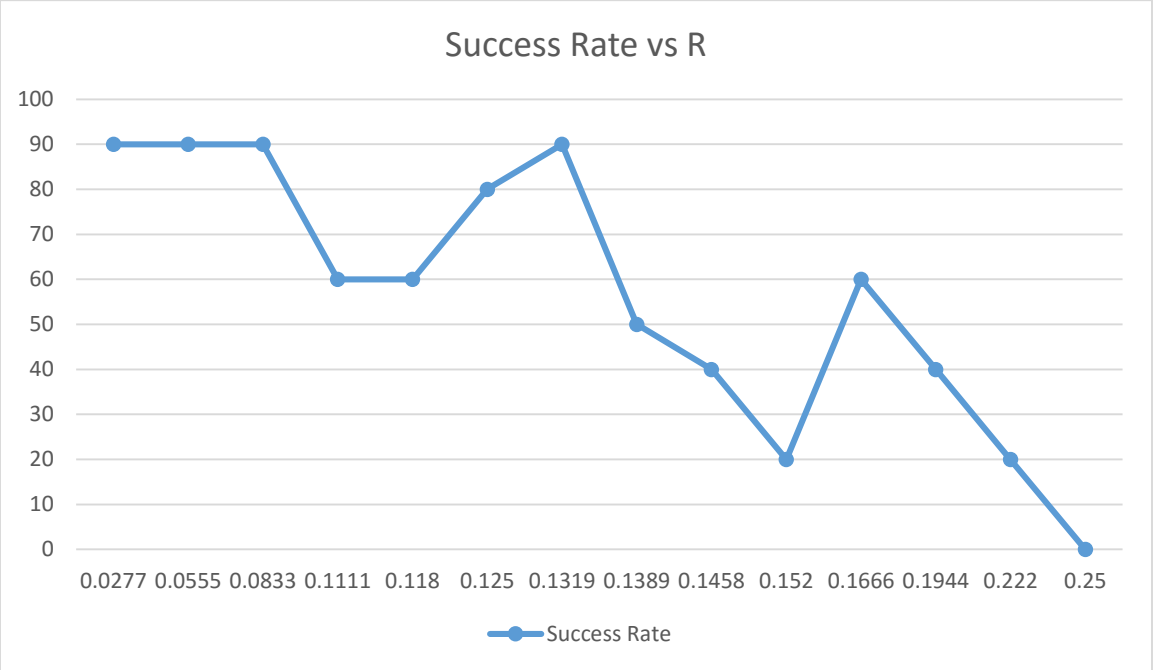
20	12	3	4	.1389	**4376.8**	**1.9288**	**1.00**	**50**
21	12	3	4	.1458	**18645.0**	**1.61**	**2.305**	**40**
22	12	3	4	.152	**380.0**	**0.256**	**0**	**20**
24	12	3	4	.1666	**27041**	**1.4**	**2.26**	**60**
28	12	3	4	.1944	**72911**	**50.5**	**78.12**	**40**
32	12	3	4	.222	**524.5**	**0.491**	**0**	**20**
36	12	3	4	.25	**0**	**0**	**0**	**0**

5.2. Find the critical value of the "hardest R" for N_largest.
[Based on your data in 5.1 above, produce a graph similar to that shown below.]



Based upon your results above, you estimate the "hardest $R_{N\_Largest}$" = .1944

5.3. How does puzzle solvability for your best combination vary with R = M / N^2? [Based on your data in 5.1 above, produce a graph similar to that shown below.]
Again, as R increases, M increase, causing my solvability % to decrease. R and the Solvability have an inverse relationship.



[For you, the x-axis will be R = M / N^2, and the y-axis will be the probability that a randomly generated standard Sudoku puzzles is solvable at all using your best combination above.]

5.4 Is the critical value for "hardest R" approximately the same as the value of R for which a random puzzle is solvable with probability 0.5?
No, the solvability for the hardest R was .40 instead of .50.

5.5 Research Question: Is hardest_$R_{largest_N}$ ≈ hardest_R_9? Why or why not? How is the hardest R related to the board parameter N? What is the shape of the function R(N), which gives you the hardest ratio for 9, 12, 15, 16, 18, 20, …, 35?

The hardest _$R_{largest_N}$ is actually lower than hardest R9. Because the larger N causes the hardest R to decrease. Therefore, the hardest R is lower for N=12 than N=9.
R(N) = M/N^2

Appendix: Extra Credit: (*) Advanced Techniques, Extra Effort, or Creativity Not Reflected Above:

[If you/your team implemented any advanced techniques, made any extra effort, or exhibited any extra creativity, then you described it

briefly but clearly above in Part 2. Here, please place any data that documents what you say you did.]

I experimented Constraint propagation with Forward Checking.

```
private boolean FC_CP(Variable v)
	{
		//Basically forward checking with constraint propagation
		ConstraintPropagation();
		for(Variable s : network.getVariables())
		{
			if(s.isAssigned())
			{
				neighbors = network.getNeighborsOfVariable(s);
				for(int i = 0; i < neighbors.size(); i++)
					if(neighbors.get(i).isAssigned())
						if(neighbors.get(i).getAssignment() ==
s.getAssignment())
							return false;
			}
		}

private boolean ConstraintPropagation()
	{
		for(Variable v : network.getVariables())
		{
			if(v.isAssigned())
			{
				neighbors = network.getNeighborsOfVariable(v);
				for(int i = 0; i < neighbors.size(); i++)
					if(!neighbors.get(i).isAssigned())

	neighbors.get(i).removeValueFromDomain(v.getAssignment());
			}
		}
		return true;
	}
```