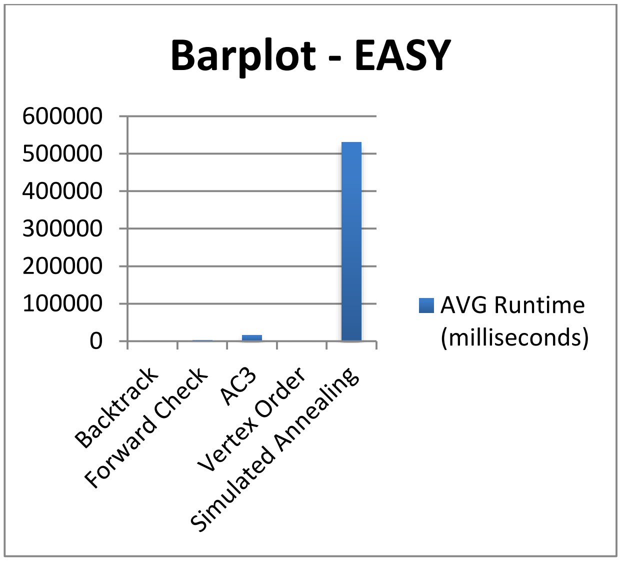
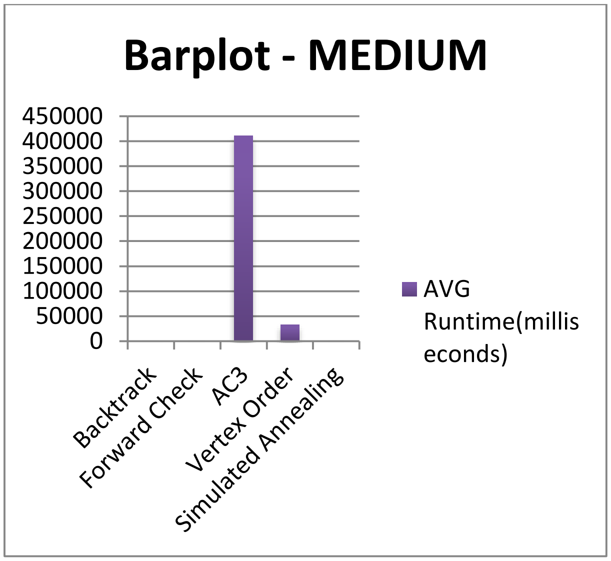
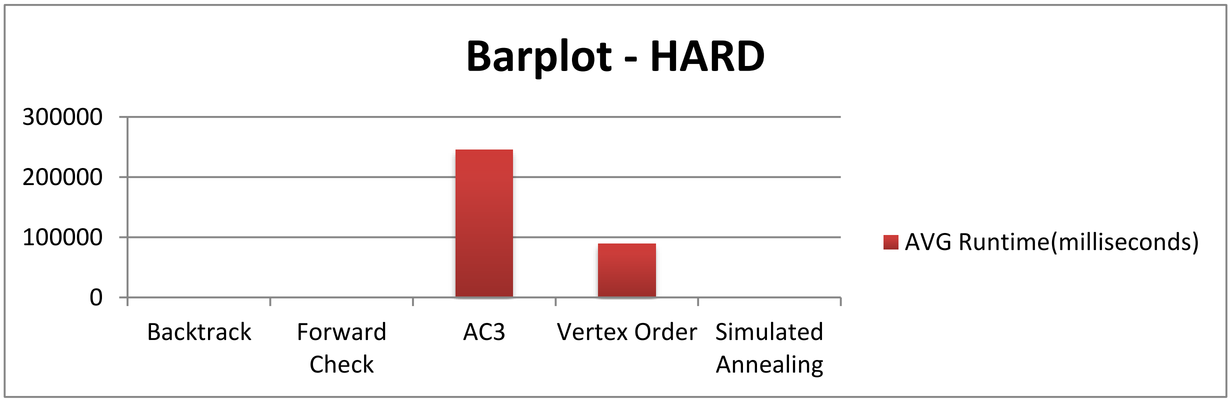
Project Report - Sudoku

Category & Statistics Algorithm	EASY			MEDIUM			HARD		
	Fast	*Slow*	*AVG*	*Fast*	*Slow*	*AVG*	*Fast*	*Slow*	*AVG*
	(in milliseconds)			*(in milliseconds)*			*(in milliseconds)*		
Backtrack	4	20	10	17	49	27	14	183	107.8
Backtrack + Forward checking	16	8504	2058.6	DNF	DNF	DNF	DNF	DNF	DNF
Arc Consistency 3	7602	21451	15602.4	350515	461756	410791	245622	245622	245622
Vertex Order	172	1160	633	10351	65208	32568.4	97386	122316	89346.4
Simulated Annealing	513074	609083	530319.2	DNF	DNF	DNF	DNF	DNF	DNF

*DNF= Did Not Finish







From the above bar-plots, it can be observed that the simulated annealing algorithm takes the highest average running time in milliseconds for an easy level sudoku puzzle, while the AC3 algorithm has the highest average runtime for a medium difficulty sudoku puzzle. Again, AC3 comparatively has the highest average runtime for sudoku puzzles of hard level.

The forward checking and simulated annealing algorithms mostly could not finish for medium and hard level puzzles i.e., time taken to find the solution for those level puzzles was way too high due to the various possibilities (beyond the system's capacity). Backtrack and vertex order approach could solve any level sudoku puzzle.

Looking into the tabulated data, we can conclude that the *backtrack algorithm* performs best as it has the least average running time (almost negligible) and it always finds a solution unlike the other four algorithms. Hence it is the best choice to use *backtrack* in order to solve any type of sudoku puzzle, be it easy, medium or hard. (i.e., most efficient algorithm)

Summarized setup employed for sudoku as a CSP :-

- ✓ Variables : Each cell in the sudoku grid/board, total 9 grids of 9 celles each= 81 variables in 9x9 board.
- ✓ Domain : Values ranging from 1-9 that could be given to each Variable. Zero(0) representing the empty squares
- ✓ Constraints : Three main constraints that decide the value assignment to the variables are - a number between 1 to 9 (or size of grid)should appear only once in every row, every column and every square/grid.

Forward checking was implemented by assigning a new value (0-9) to a variable and backtracking to check for any inconsistencies of this new assignment with every row, column and grid (neighboring domains) respectively. Hence, theoretically it is to be faster than the normal Backtrack algorithm as the discovered inconsistencies are usually not revisited. But, what it really did was not according to theory. There were too many possibilities for every variable, that for medium and hard sudoku puzzles, the program did not finish or find a solution. It was beyond the system's capacity to do so.

The Arc consistency- version 3 (ac3) algorithm helped resolve this issue. It cut down the various possibilities to a few (within system's capacity) by checking and queuing only the consistent arcs between the variable domains. Each of these arcs are later popped out of the queue, backtrack is used to make the right guess and the queue is updated again. This method thereby helped the program terminate, that is, it actually found a solution to all three levels of sudoku , though the running time was higher than the normal backtrack solution.

For simulated annealing, the system was basically setup as a simulation machine with a best solution score of -162 along with a temperature regime (system's initial temperature) which is inversely proportional to the number of iterations. Moves implemented were to randomly fill up an incomplete sub-square after which another random sub-square with two values flipped is filled. Each of their result scores are calculated and compared (Boltzmann formula used to decide if new-score > old-score) and this repeats or it is re-flipped. The temperature regime of 2.5 was a tough decision accomplished after referring to a few papers and Ben-Ameur iterative method.

Backtrack guesses the numbers to be filled while the other algorithms look for a definite answer that is time consuming. The number of cases that had to be considered were vast and the tree structure got too big for the system to handle. If possible, to make the solving better by even the best performer, an AI that could guess like humans could be incorporated . This is because guessing gets challenging as the level of complexity increases. The AI could be designed to never lose and always find the solution given any level of complexity of the sudoku puzzle. This would be an excellent future enhancement for this project's improvement.

Table 1 - Sudoku EASY

Algorithm(AVG Runtime (in ms)) Initial configuration	Backtrack	Backtrack + Forward Check	Arc Consistency	Vertex Order	Simulated Annealing
0 3 0 1 5 0 0 2 0 0 0 0 0 2 8 0 0 3 8 0 2 0 0 0 7 4 0 9 7 0 0 4 0 0 0 8 0 0 1 5 0 0 9 6 0 0 0 5 6 8 0 0 0 1 0 0 8 4 0 5 0 0 0 5 0 0 3 1 0 4 0 0 4 6 0 0 0 0 0 1 9	20	158	7602	340	513074
0 0 6 0 0 0 3 0 0 2 7 3 0 0 4 5 0 0 5 0 0 9 0 0 0 2 4 0 0 0 3 0 0 0 0 5 0 0 8 0 2 0 6 3 0 0 3 0 6 8 5 0 0 2 0 9 0 0 4 7 0 6 8 8 2 0 0 0 3 9 1 0 6 0 0 0 9 0 0 0 0	4	1448	21451	1160	609083
6 0 0 8 0 0 0 0 0 0 3 2 0 0 6 1 0 4 0 0 7 0 1 2 6 9 0 0 0 6 0 4 3 0 0 7 0 0 0 0 0 1 0 2 3 3 0 8 0 2 9 0 5 0 0 0 9 0 0 0 0 8 1 0 7 5 0 0 8 0 6 0 0 8 3 2 0 5 0 0 9	4	167	14245	172	460165
2 9 5 3 7 0 0 0 0 0 0 3 0 0 0 6 0 0 0 0 0 9 8 0 2 0 3 0 0 0 1 0 0 0 2 0 3 0 6 7 2 5 0 0 0 7 0 8 0 0 0 0 3 1 0 3 0 6 0 0 9 0 0 5 6 7 2 9 3 1 4 0 0 0 1 0 0 7 0 6 0	12	16	17347	682	520138
0 0 7 6 0 0 9 0 2 4 0 8 1 2 9 0 0 0 0 9 6 0 0 3 0 1 0 0 6 1 0 8 0 2 0 0 0 8 0 2 0 0 0 5 0 3 2 0 0 0 5 0 0 8 6 0 5 0 0 0 3 2 1 9 4 0 3 0 0 7 0 0 8 0 0 0 6 2 0 9 0	10	8504	17367	811	549136

Table 2 - Sudoku MEDIUM

Algorithm(AVG Runtime (in ms)) **Initial configuration**	**Backtrack**	**Backtrack + Forward Check**	**Arc Consistency**	**Vertex Order**	**Simulated Annealing**
0 0 7 0 0 0 6 0 0 0 5 1 0 0 8 2 0 0 6 0 0 0 0 0 0 0 0 0 2 0 3 0 7 0 4 0 3 0 0 2 4 0 0 0 7 0 9 0 0 0 0 0 0 8 0 0 6 0 2 0 8 0 9 0 0 2 0 0 0 1 0 0 0 0 0 0 0 0 0 3 0	17	DNF	420102	17999	DNF
0 0 2 5 0 7 0 0 3 0 0 0 0 0 0 4 0 6 1 0 6 0 0 0 0 0 0 0 0 0 6 3 9 0 0 0 9 0 0 0 0 8 0 0 7 2 4 0 0 0 0 0 0 0 0 0 9 4 0 0 7 0 0 0 0 0 0 0 0 2 5 0 0 0 0 3 9 5 0 0 0	20	DNF	461756	19841	DNF
0 0 0 0 0 5 6 0 2 0 4 0 0 6 0 7 0 0 0 0 3 0 0 0 0 0 0 0 0 7 0 0 0 0 5 6 0 9 6 0 7 0 0 3 0 0 2 0 0 9 6 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 8 0 7 0 0 5 0 0 2 3 0 0 0 4 0	17	DNF	350515	10351	DNF
0 0 0 7 4 0 2 0 0 8 0 0 0 0 0 0 0 0 0 1 0 0 8 0 0 0 5 6 3 0 0 0 7 0 0 8 1 0 0 3 0 6 0 0 0 0 7 0 0 0 0 3 0 2 0 8 0 0 0 0 0 0 0 5 0 0 0 0 3 0 7 0 0 0 0 2 0 0 1 3 0	32	DNF	DNF	65208	DNF
0 0 7 6 0 0 0 0 0 0 0 8 5 1 0 6 0 0 9 0 0 7 0 0 0 0 2 0 0 0 0 0 0 0 0 0 1 0 0 0 0 2 5 0 0 0 0 0 0 0 3 1 8 4 0 4 0 0 9 0 0 0 1 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 8 6 7	49	DNF	DNF	49443	DNF

Table 3 - Sudoku HARD

Algorithm(AVG Runtime (in ms)) **Initial configuration**	**Backtrack**	**Backtrack + Forward Check**	**Arc Consistency**	**Vertex Order**	**Simulated Annealing**
0 0 0 0 1 0 7 0 0 0 5 0 0 0 0 0 9 0 1 0 0 0 0 8 0 0 0 0 0 3 1 9 0 0 0 0 0 8 0 0 0 2 0 7 0 7 0 2 0 0 0 5 0 0 8 0 0 0 0 0 3 0 5 0 6 0 0 0 0 0 4 0 0 0 0 3 0 6 0 0 1	183	DNF	DNF	117248	DNF
0 0 0 0 0 6 0 9 0 0 2 0 4 0 0 0 0 3 5 0 0 9 1 0 7 0 0 0 0 8 0 0 0 0 0 2 0 4 0 0 0 0 0 0 6 0 3 0 0 0 0 9 0 0 7 0 0 0 0 1 0 0 0 0 5 0 0 2 0 0 0 9 0 0 9 6 7 0 0 3 0	99	DNF	DNF	97386	DNF
0 5 0 0 0 0 0 6 0 0 0 0 3 0 0 8 0 0 0 2 1 4 5 0 0 0 0 1 0 0 0 7 0 3 0 6 0 7 0 0 0 5 0 0 0 0 0 0 0 0 4 1 0 0 7 0 0 0 0 0 0 4 0 0 0 2 0 0 6 0 3 9 8 0 0 2 0 0 0 0 0	14	DNF	245622	10919	DNF
0 7 0 0 0 0 0 0 5 0 0 3 2 0 0 6 0 0 1 0 0 0 0 9 7 8 0 7 0 0 3 0 0 0 0 0 0 0 2 0 4 0 0 0 0 0 0 5 6 0 0 0 0 0 0 0 7 9 0 0 0 2 0 0 3 0 0 7 0 5 1 0 0 0 0 0 0 1 0 0 8	183	DNF	DNF	122316	DNF
0 0 0 0 0 0 7 3 0 0 0 0 0 4 0 0 0 1 0 0 9 7 0 2 5 0 0 5 0 0 0 0 0 9 0 0 0 0 0 0 8 4 0 0 0 0 0 0 0 2 1 0 0 0 8 0 0 6 1 0 0 4 0 0 5 0 0 0 9 0 0 0 3 0 2 0 0 0 0 0 0	60	DNF	DNF	98863	DNF