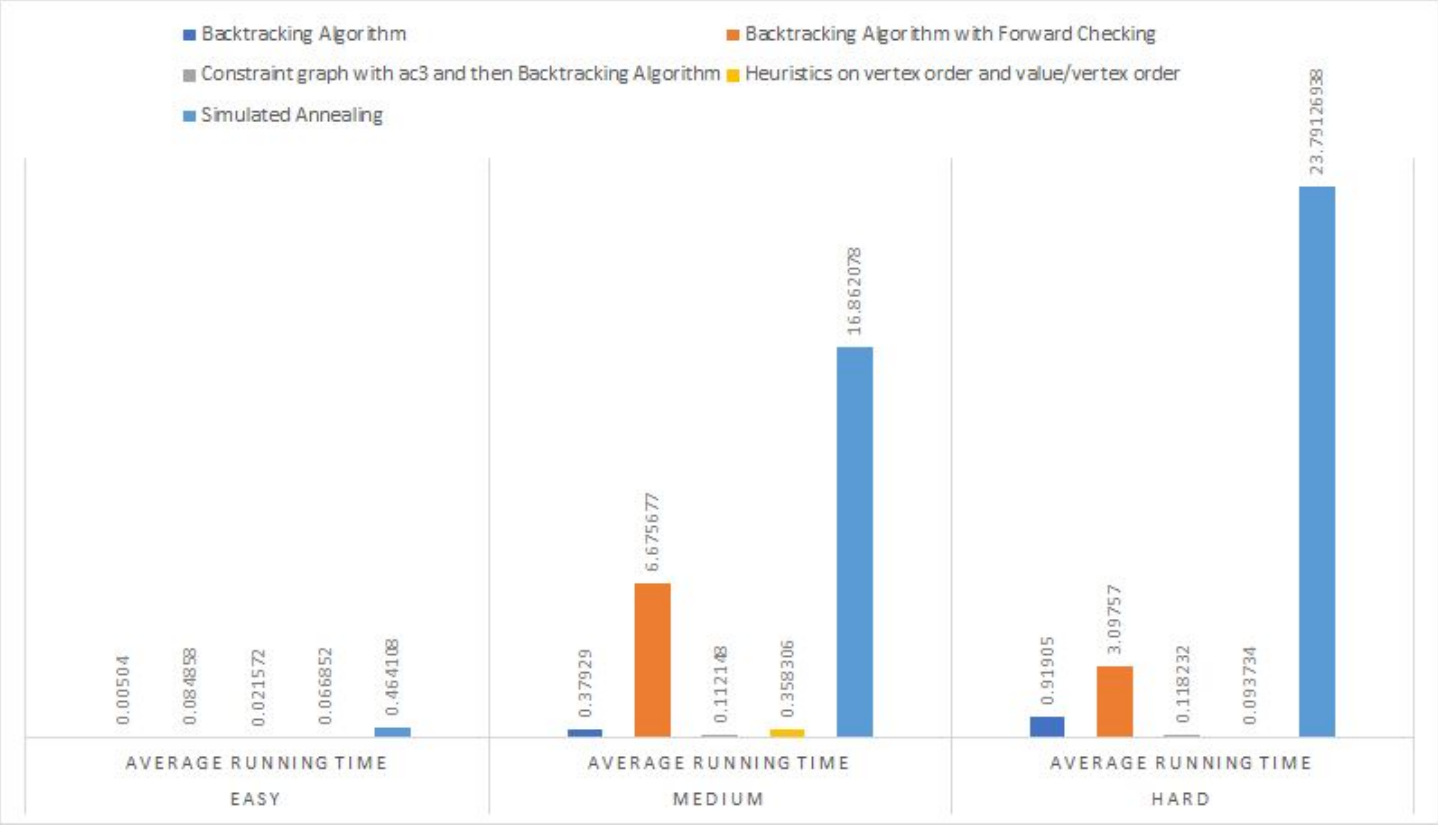
CS 580: Final Project Report

Tabulated Statistics:

Approach	Easy(Time in seconds)			Medium(Time in seconds)			Hard(Time in seconds)		
	Fastest Running Time	Slowest Running Time	Average Running Time	Fastest Running Time	Slowest Running Time	Average Running Time	Fastest Running Time	Slowest Running Time	Average Running Time
Backtracking Algorithm	0.00077	0.01008	0.00504	0.01785	1.51859	0.37929	0.00432	1.77419	0.91905
Backtracking Algorithm with Forward Checking	0.05719	0.11146	0.084858	2.75142	10.41411	6.67568	0.14655	8.77724	3.09757
Constraint graph with AC-3 and then Backtracking Algorithm	0.01619	0.02440	0.02157	0.04096	0.20255	0.11215	0.03038	0.26125	0.11823
Heuristics on vertex order and value/vertex order	0.06207	0.07108	0.066852	0.13631	0.72936	0.358306	0.07165	0.15621	0.093734
Simulated Annealing	0.18619	1.20946	0.46411	8.82725	55.58330	17.25304	10.65151	45.13951	23.79127

Average Running Time for each Algorithm:



Best Algorithm:

Adding **LCV and MRV Heuristics** to the traditional backtracking improves the performance.

In most cases **Simulated Annealing** either runs very slow and may find a solution or runs fast and does not end up finding a goal at all. As, our algorithm for SA implements a very gradual change in temperature, it does solve most of the sudoku problems in one run but, takes more time.

Backtracking with **Forward Checking** should ideally run faster than traditional backtracking. But our implementation of the algorithm creates a new board for each iteration instead of the applying the changes to the current board; spending more time and memory. Thus, implementation complexities have caused the algorithm to run slower than backtracking itself.

According to the statistics, **AC-3** performs the best, because everytime the domain of a cell is reduced, AC-3 propagates a series of further checks on the previously reduced domains.

Looking at the generated results AC-3 performs best in all the cases. Thus, it should be used to solve the sudoku puzzle generally.

Sudoku as CSP:

Variables: All the cells in a sudoku puzzle, i.e. X_{ij} where i,j take values from 1 to 9

Domains: Domain is the set of all possible values that each cell from X_{11}-X_{99} can take, i.e. $X_{ij} \in \{1,2,...,8,9\}$

Constraints: Each cell will be constrained by every other cell in its corresponding row, column and block. Thus, for every cell we get 8 row constraints + 8 cell constraints + 8 block constraints

1. Row constraint : $X_{11} \neq X_{12},..., X_{11} \neq X_{19}$
2. Column constraint : $X_{11} \neq X_{21},..., X_{11} \neq X_{91}$
3. Block constraint: $X_{11} \neq X_{12},..., X_{11} \neq X_{33}$

```
   1 2 3  4 5 6  7 8 9
 -----------------------
1 | 0 3 0 | 2 0 0 | 0 0 0 |
2 | 0 7 0 | 0 0 0 | 0 0 4 |
3 | 8 0 1 | 0 7 9 | 6 2 0 |
 -----------------------
4 | 0 0 8 | 0 5 3 | 9 4 0 |
5 | 0 0 0 | 0 2 4 | 0 8 1 |
6 | 2 5 4 | 0 0 0 | 0 7 0 |
 -----------------------
7 | 0 0 0 | 9 0 2 | 4 6 0 |
```

Forward Checking Implementation:

The board is implemented as a list of lists [9 rows x 9 cells]. To save the possible domain values of each cell we have used a 'forwardBoard' which is a list of lists [81 cells x 9 domain values]. The 'forwardBoard' is initialized such that it does not conflict with the values that are pre-assigned in the sudoku problem. Next, we choose an empty cell sequentially and randomly pick a domain value for it by a lookup in the 'fowardBoard'. Then we check if the value satisfies the constraints of sudoku, if it does, we assign the value to the cell and repeat the procedure for the next empty cell. If the value does not satisfy the constraints we pick another value randomly. If none of the values satisfy the constraints we clear the assignment and backtrack to the previous satisfied state.

How AC-3 helped:

AC-3 maintains three lists viz. variables, domains and constraints. Constraints is a dictionary which contains pairs of: variables and a list of variables which it constraints.

$C = \{(X_{11},\{X_{12},\ldots, X_{19}, X_{21},..X_{91},.., X_{33}\}), ... \}$

We have assigned {1,..,9} as a domain to each cell. So, AC-3 is called initially to reduce the domain for each cell based on the constraints. We then call the backtracking algorithm with AC-3 as its inference.

First, we sequentially pick a cell for assignment and then sequentially pick a value to assign the cell. After the assignment, AC-3 is called to reduce the domain values for each cell. This call return two values, first, a boolean value representing whether the domain was reduced and the second the list of the domain values that were removed. If the domain was reduced, we repeat this procedure for the next empty cell, else, we add the previously removed domain values back to its set of domains.

Backtracking calls AC-3 function with a list of all the constraints associated with the just assigned cell. This assignment is evaluated against the constraints of the cell. If any of the domain values of the constraints conflict with this assignment, the domain of the constraints gets reduced. This reduction triggers a repetitive check on all the cells that constrained with the cell whose domain was reduced.

If the domain of values reduces, AC-3 returns true and the list of conflicting values from its domain.

Everytime the domain of a cell is reduced, AC-3 propagates a series of further checks on these reduced domains. This makes AC-3 is a powerful Inference mechanism.

Simulated Annealing:

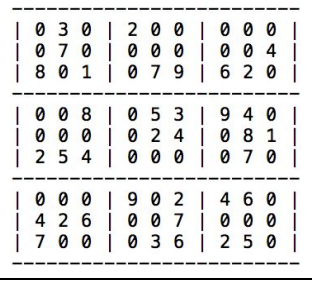
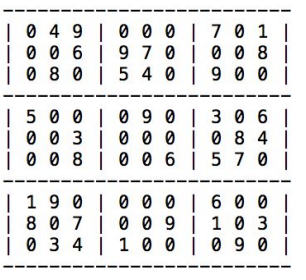
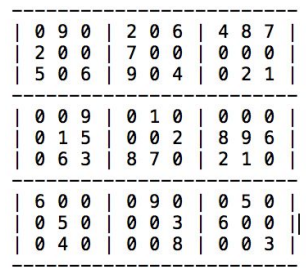
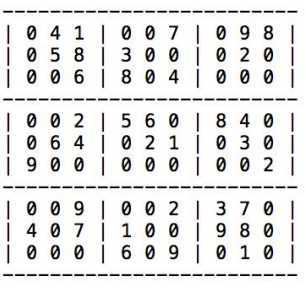
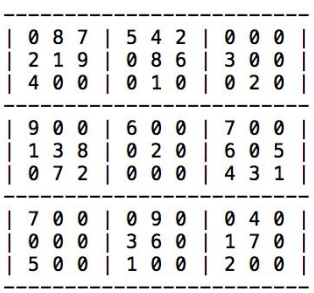
This algorithm randomly fills out the cells in each of the 9 blocks to be consistent solutions for each block. It then calculates the score of a board configuration by giving -1 for every unique assignment in each row, column, or block. Thus, best solution will have a cost of -81 + -81 = -162. This also becomes our stopping criteria.

Then we do 400000 iterations and generate a neighbour. Neighbour for the current puzzle is generated by the 'make_neighborBoard()' function by randomly selecting a block, then randomly swapping two of its assigned entries and then its score is evaluated. The difference between the the scores of neighbour and current puzzle is stored in the variable called 'delta'. The temperature 'T' is initialised to 0.5 and decreases geometrically as T = .99999*T in each iteration. We only make the neighbour as the current board only if exp(delta/T) > rand(0,1). Since the algorithm also allows bad moves, we must decrease the temperature gradually to have a chance of getting a good move and ultimately reaching the best solution. Hence, the decreasing factor.

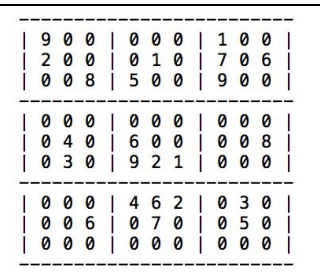
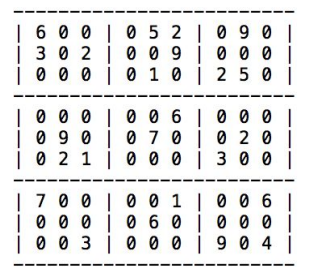
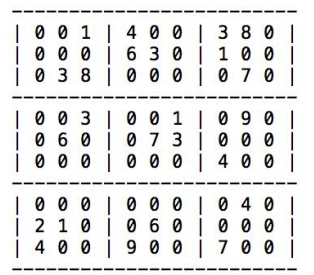
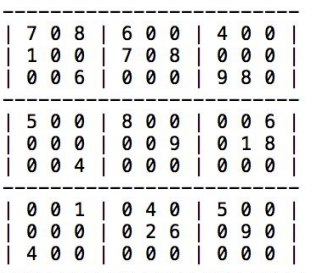
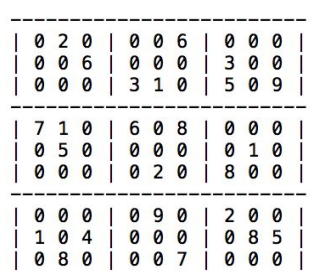
Improvement for best performer AC3:

Instead of sequentially picking a cell for assignment and then sequentially picking a value to assign the cell, we can use a heuristic function to choose the next cell to be assigned and the value to assign. We can use the Minimum Remaining Value heuristic to choose the next cell to be assigned and the Least Constraining Value heuristic to choose a value to assign the cell.

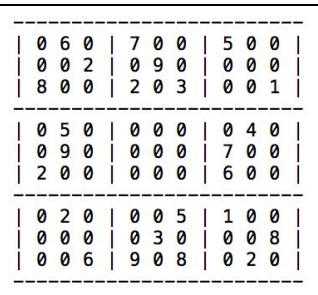
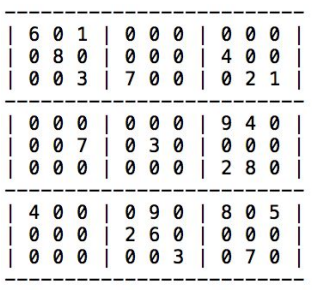
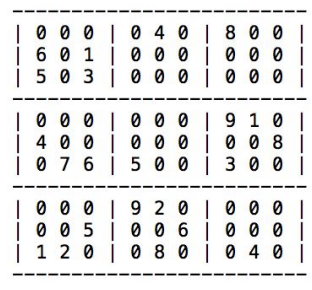
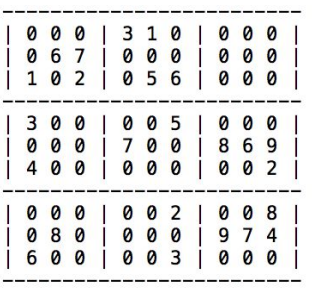
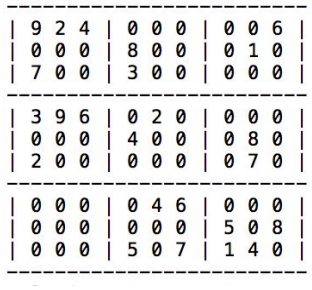
EASY PUZZLES

INITIAL CONFIGURATION	APPROACH	RUNNING TIME
Puzzle 1		
	Backtracking Algorithm	0.00915
	Backtracking Algorithm with Forward Checking	0.05719
	Constraint graph with ac3 and then Backtracking Algorithm	0.02440
	Heuristics on vertex order and value/vertex order	0.07108
	Simulated Annealing	1.20946
Puzzle 2		
	Backtracking Algorithm	0.01008
	Backtracking Algorithm with Forward Checking	0.09986
	Constraint graph with ac3 and then Backtracking Algorithm	0.02172
	Heuristics on vertex order and value/vertex order	0.06980
	Simulated Annealing	0.24903
Puzzle 3		
	Backtracking Algorithm	0.00188
	Backtracking Algorithm with Forward Checking	0.09098
	Constraint graph with ac3 and then Backtracking Algorithm	0.01619
	Heuristics on vertex order and value/vertex order	0.06664
	Simulated Annealing	0.24370
Puzzle 4		
	Backtracking Algorithm	0.00334
	Backtracking Algorithm with Forward Checking	0.11146
	Constraint graph with ac3 and then Backtracking Algorithm	0.02200
	Heuristics on vertex order and value/vertex order	0.06467
	Simulated Annealing	0.43216
Puzzle 5		
	Backtracking Algorithm	0.00077
	Backtracking Algorithm with Forward Checking	0.06480
	Constraint graph with ac3 and then Backtracking Algorithm	0.02355
	Heuristics on vertex order and value/vertex order	0.06207
	Simulated Annealing	0.18619

MEDIUM PUZZLES

INITIAL CONFIGURATION	APPROACH	RUNNING TIME
Puzzle 6		
	Backtracking Algorithm	0.16348
	Backtracking Algorithm with Forward Checking	10.41411
	Constraint graph with ac3 and then Backtracking Algorithm	0.06528
	Heuristics on vertex order and value/vertex order	0.72936
	Simulated Annealing	55.5833
Puzzle 7		
	Backtracking Algorithm	1.51859
	Backtracking Algorithm with Forward Checking	3.63940
	Constraint graph with ac3 and then Backtracking Algorithm	0.20255
	Heuristics on vertex order and value/vertex order	0.20532
	Simulated Annealing	8.82725
Puzzle 8		
	Backtracking Algorithm	0.10857
	Backtracking Algorithm with Forward Checking	7.63481
	Constraint graph with ac3 and then Backtracking Algorithm	0.1353
	Heuristics on vertex order and value/vertex order	0.33307
	Simulated Annealing	10.85978
Puzzle 9		
	Backtracking Algorithm	0.08797
	Backtracking Algorithm with Forward Checking	8.93869
	Constraint graph with ac3 and then Backtracking Algorithm	0.11665
	Heuristics on vertex order and value/vertex order	0.38747
	Simulated Annealing	3.03523
Puzzle 10		
	Backtracking Algorithm	0.01785
	Backtracking Algorithm with Forward Checking	2.75142
	Constraint graph with ac3 and then Backtracking Algorithm	0.04096
	Heuristics on vertex order and value/vertex order	0.13631
	Simulated Annealing	7.95961

HARD PUZZLES

INITIAL CONFIGURATION	APPROACH	RUNNING TIME
Puzzle 11		
	Backtracking Algorithm	1.77419
	Backtracking Algorithm with Forward Checking	0.146551
	Constraint graph with ac3 and then Backtracking Algorithm	0.26125
	Heuristics on vertex order and value/vertex order	0.15621
	Simulated Annealing	13.67586
Puzzle 12		
	Backtracking Algorithm	1.28160
	Backtracking Algorithm with Forward Checking	8.77724
	Constraint graph with ac3 and then Backtracking Algorithm	0.05499
	Heuristics on vertex order and value/vertex order	0.08432
	Simulated Annealing	37.6946
Puzzle 13		
	Backtracking Algorithm	0.59825
	Backtracking Algorithm with Forward Checking	1.93944
	Constraint graph with ac3 and then Backtracking Algorithm	0.21097
	Heuristics on vertex order and value/vertex order	0.08406
	Simulated Annealing	45.13951
Puzzle 14		
	Backtracking Algorithm	0.93687
	Backtracking Algorithm with Forward Checking	0.43869
	Constraint graph with ac3 and then Backtracking Algorithm	0.03357
	Heuristics on vertex order and value/vertex order	0.07165
	Simulated Annealing	10.65151
Puzzle 15		
	Backtracking Algorithm	0.00432
	Backtracking Algorithm with Forward Checking	4.18594
	Constraint graph with ac3 and then Backtracking Algorithm	0.03038
	Heuristics on vertex order and value/vertex order	0.07243
	Simulated Annealing	11.79487

Note: All time in seconds.