Name: Manva Patel

Email address: m329pate@uwaterloo.ca

Student number: 20815074

ASSIGNMENT 1: QUESTION 1

Representation of Problem:

1) States and their representation: Let n be the number of cities in a given TSP problem and 0<=x<=n. Then a state at a given time is:
   - a path from A to any other city, having x number of cities in between
2) Initial state: A
3) Goal state: Start from A, visit every citiy once and come back to A. Choose a path such that the total cost generated is lowest of all possible costs.
4) Operators: Choose all cities connected to the current city. Select a state such that the f(n) value of the path is minimum of all the other states calculated before.
5) Cost: g(n) + h(n)
   g(n): Euclidean distance from A to current city.
   h(n): The heuristic value of current city.(Estimated distance from current node to end node)

Heuristic Function:

-The heuristic function used in solving TSP is Minimum Span Tree(MST).

-A MST of a graph is a subgraph that connects all vertices in the graph with a minimum total weight for the edges.

-In this case we take a node whose heuristic is to be calculated as input and using MST, we get the minimum cost of going from current node to A(goal node), traversing nodes which are not in the path of current node.

-We use Prim's algorithm to implement MST. Using Prim's algorithm, the heuristic we obtain is admissible because it gives us the path having minimum cost for going from current node to A. If some other path is chosen for going to A, then the cost will always be higher than the one we have calculated. So the heuristic value is optimistic, which means that the heuristic is admissible.

-For example, consider a 5 city problem. We are at a node C and it has covered the path A->D->C. Now we have to find the heuristic value of C.

For that we find MST(C,A) excluding the node D. Suppose the path it returns for travelling from C to A is C->E->B->A. Then this path is the minimum cost it takes in travelling from C to A through all remaining cities.

So if we take any other path, the cost(C,A)>MST(C,A) i.e., the cost of going from C to A will always be more than the cost calculated by MST.

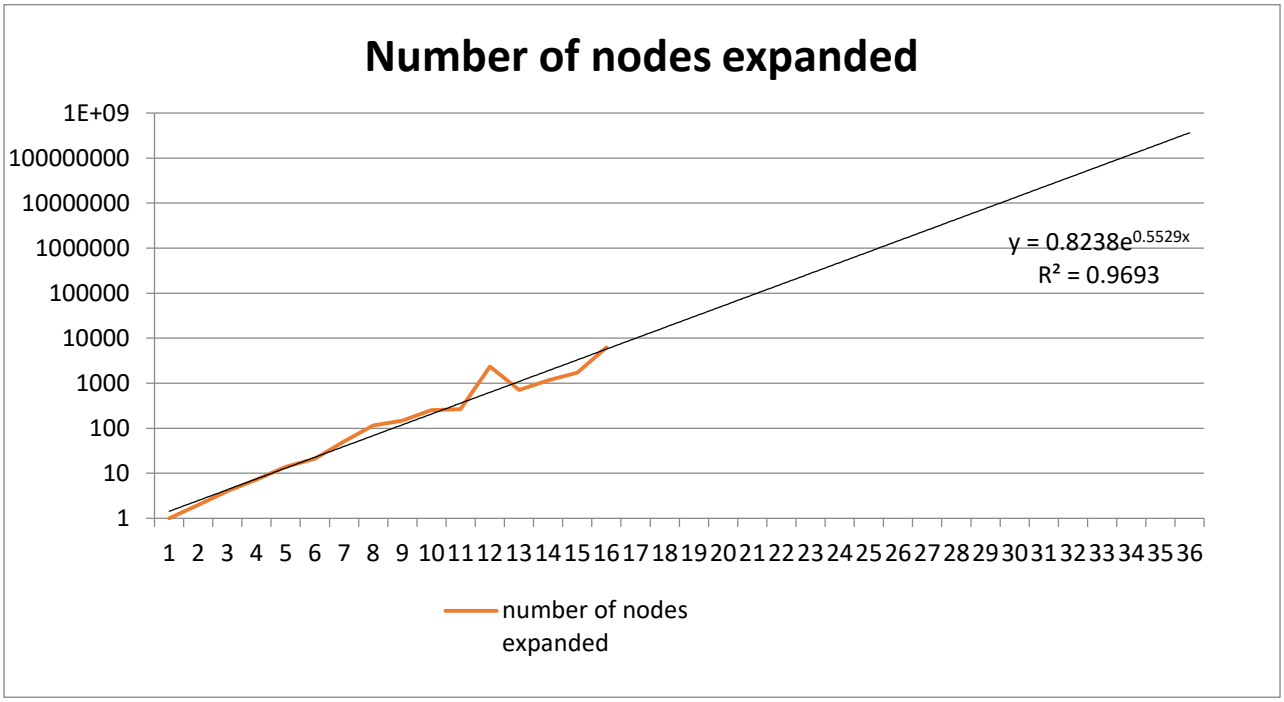
-Hence, MST is admissible heuristic

Results for h(n) = MST:

-Given below is a table showing the average number of nodes expanded for each city n, where 0<n<=16 when the Minimum Span Tree heuristic is applied to the nodes

Number of cities	Number of nodes expanded
1	1
2	2
3	4
4	7.3
5	13.9
6	21.2
7	50.6
8	115.6
9	147.5
10	253.7
11	265.8
12	2332.2
13	704.2
14	1153.7
15	1725.9
16	6273.5

-The graph plotted for these values is given below



Graph 1: Plots the number of nodes expanded for given number of cities where h(n) = MST. The x-axis shows the number of cities and y-axis shows the number of nodes expanded.

-As seen from the graph, when we extrapolate the graph, the equation we get is y = .8238e^0.5529x, where x=no of cities.

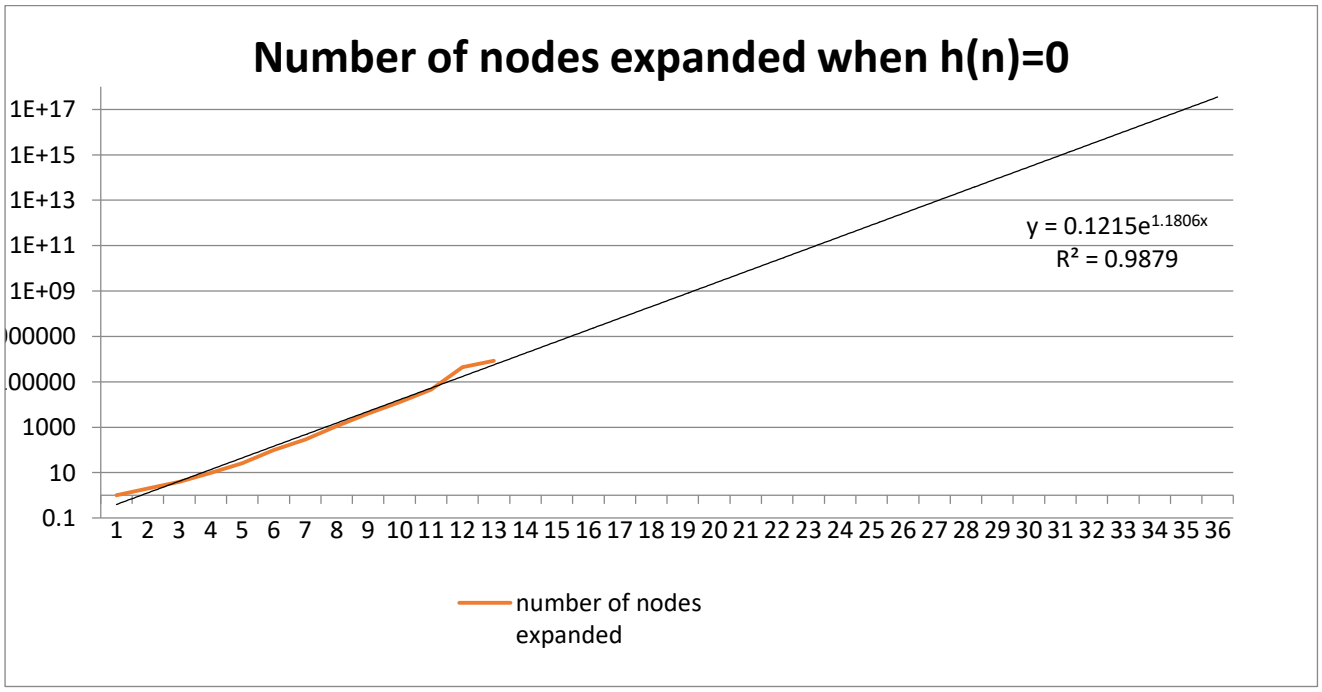
-So for n=36, according to the obtained equation, we get the value of y as 361,644,594.

-**Results for h(n) = 0:**

-Given below is a table showing the average number of nodes expanded for each city n, where 0<n<=16 when heuristics are not applied to the nodes. For n>13, the program took more than 5 minutes and did not terminate.

Number of cities	Number of nodes expanded
1	1
2	2
3	4
4	9.8
5	26.2
6	96.9
7	287.3
8	1131.2
9	3991.4
10	13006.8
11	46115.5
12	437213.3
13	848601.2

-The graph plotted for these values is given below



Graph 2: Plots the number of nodes expanded for given number of cities where h(n) = 0. The x-axis shows the number of cities and y-axis has the logarithmic scale.

-As seen from the graph, when we extrapolate the graph, the equation we get is y = .1215e^1.1806x, where x=no of cities.

-So for n=36, according to the obtained equation, we get the value of y as 3.484e17

Difference in the performance of the two different heuristics:

-As seen from the tables, the number of nodes expanded when we use MST as heuristic is quite less than the number of nodes expanded without using the heuristics.

-Also, when the number of cities increases, there is a drastic increase in the number of nodes traversed when the heuristic is 0 as compared to when it is not 0. When we solve TSP without using heuristics, it solves upto 9 cities in less time. But when the number of cities becomes greater than 9, the time taken to solve TSP increases(takes about 2-5 minutes). And if the number of nodes exceed 13, then the program does not terminate. Whereas when we use heuristics, the time taken to solve 16 nodes is under 2 minutes only.

-This is because the heuristic adds the probable cost of going from a node n back to A. It could be possible that the cost of travelling to a node n is less but the cost of traversing the rest of the nodes back to A is very high.

-If we are using heuristics, then the h(n) would be very high and so the resulting f(n) would also be high. So, the algorithm would not choose that path even if it's g(n) is low.

-Whereas if we do not have heuristics, then the algorithm chooses the node and then after expanding its children, it will have to traverse back. Thus it will generate more number of nodes.

-From this, we can conclude that the better the heuristic, the less number of nodes will be generated.

-We can manage to solve the 36 city problem if we have a better heuristic. Using MST, we can see that the number of nodes expanded for 36 cities is 361,644,594. While this is a large number, and would take a lot of time, it is still achievable with the help of a better heuristic. Whereas if we don't use heuristics, then we would have to traverse 3.484e17 nodes, which is not possible.

Conclusion:

-Using MST as a heuristic solves the 16 city problem in less than 2 minutes. Also we can say that the heuristic is admissible. It reduces the number of nodes to be traversed drastically. Whereas, while it does not solve the 36 city problem in less than 5 minutes, the number of nodes that have to be traversed to solve it is quite less as compared to not using a heuristic.

ASSIGNMENT 1: QUESTION 2

Sudoku as a CSP:

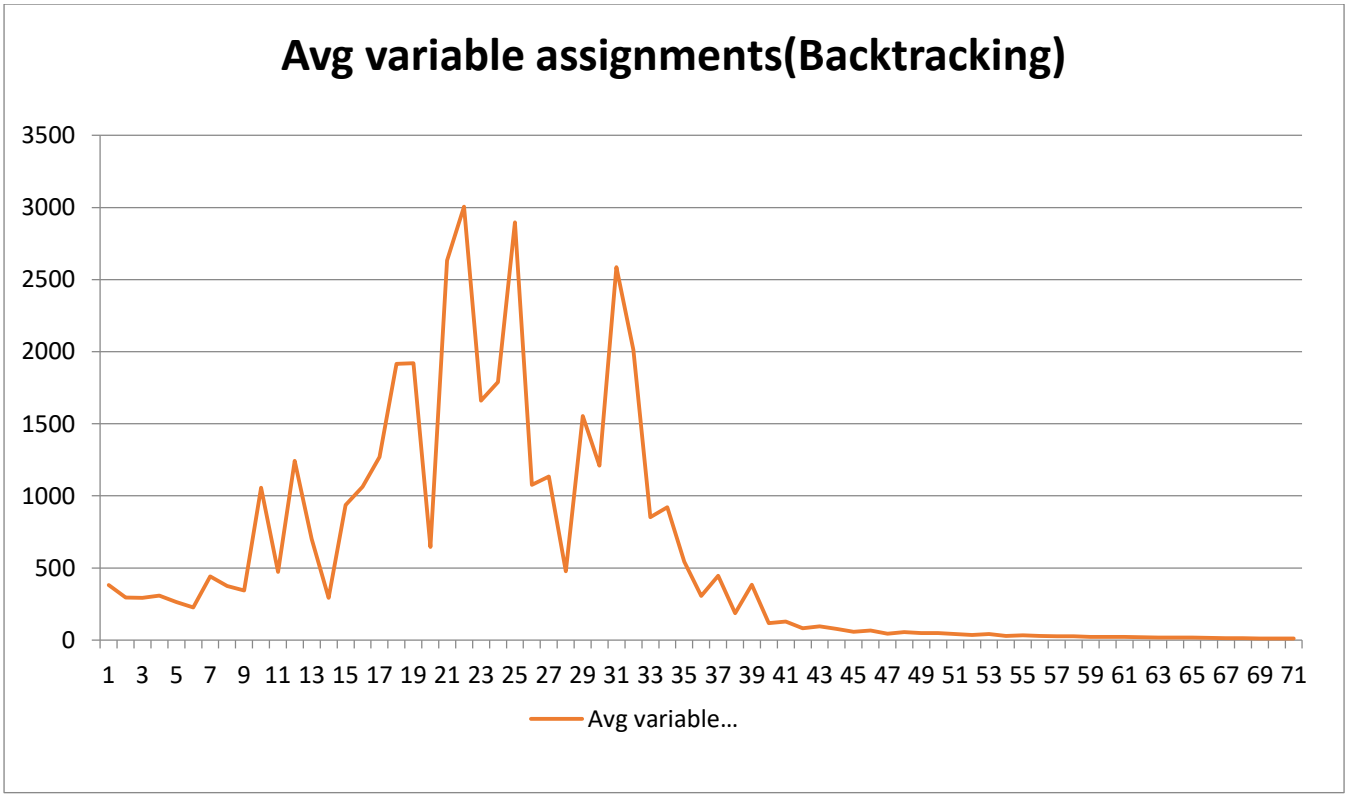
1) Variables: X_{ij} , where 0<= i, j <9
2) Domain : {1,2,,3,4,5,6,7,8,9}
3) Constraints:
    - No cells in the same row can have same value :
      Value(X_{ij}) != {Value(X_{i0}), Value(X_{i1}), ….Value(X_{i8})}
    - No cells in the same column have same value:
      Value(X_{ij}) != {Value(X_{0j}), Value(X_{1j}), ….Value(X_{8j})}
    - No cells in the same sub-square have the same value

-The variables of a Sudoku would be all the squares available in the 9x9 grid

-The domain values of a specific variable would be any number from 1 to 9.

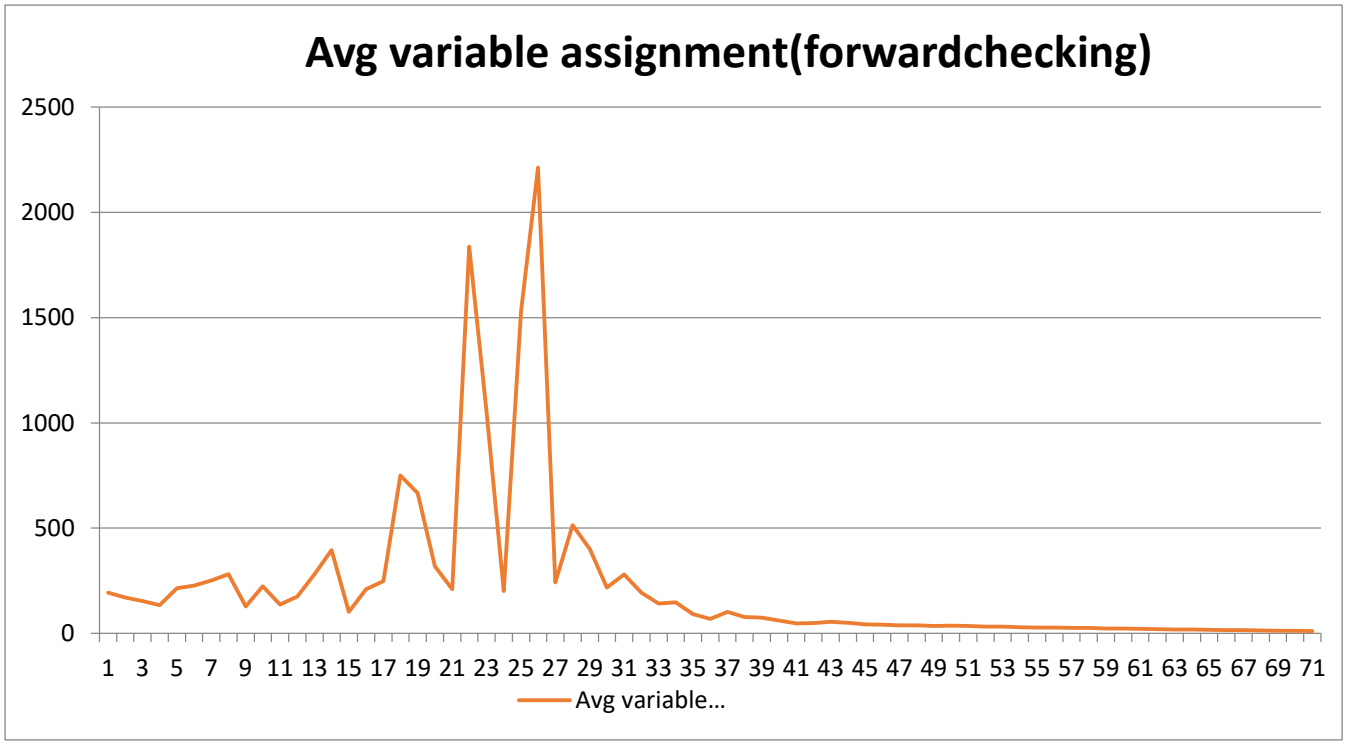
-The constraints of a Sudoku are that variables in the same row, column and 3x3 sub-square should not have the same value

Graph for solving Sudoku using Backtracking:



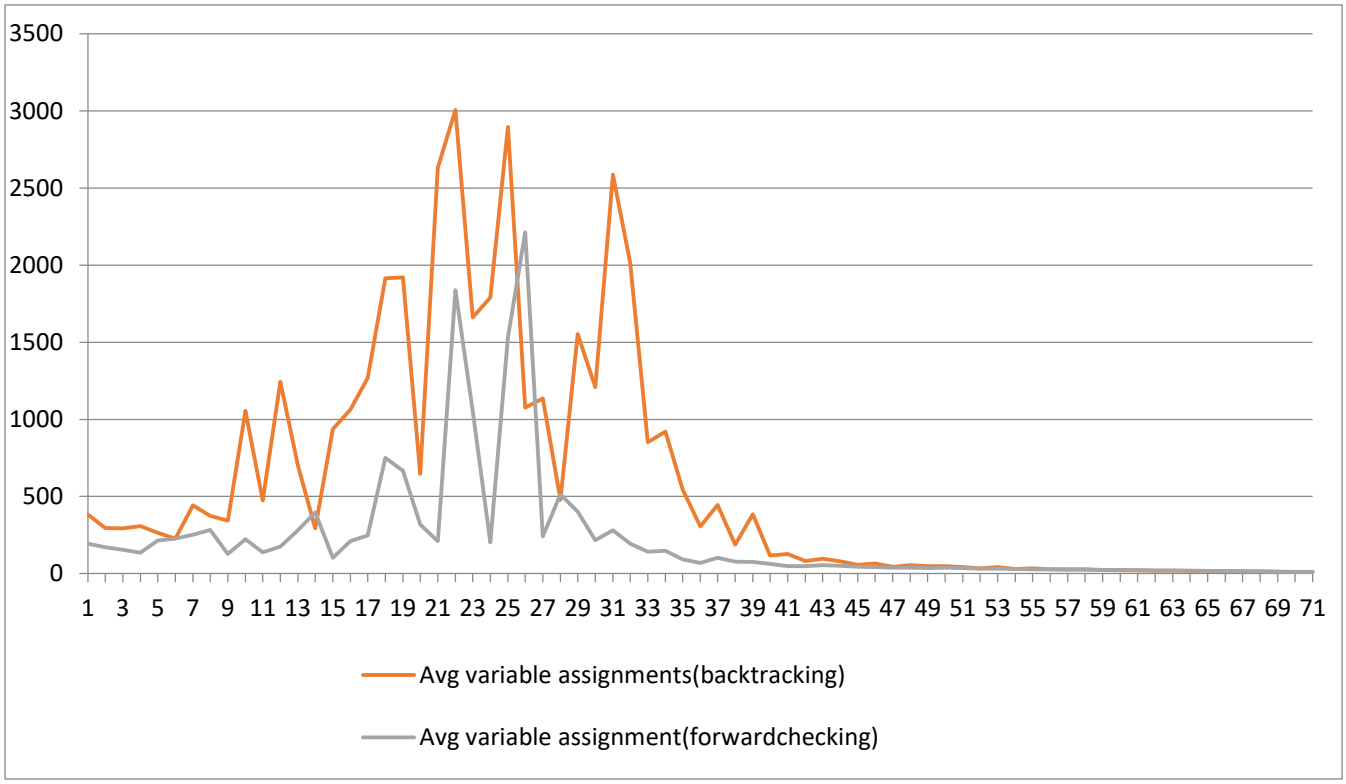
Graph 1: x-axis shows the number of initial values given. Y-axis shows the number of average variable assignments done for each initial value, when the system is using backtracking

Graph for solving Sudoku using backtracking and forward checking:



Graph 1: x-axis shows the number of initial values given. Y-axis shows the number of average variable assignments done for each initial value, when the system is using backtracking + forward checking

Comparison of both the graphs:



-As seen from the above graphs, we can infer when using backtracking, that the number of variables assigned will be the highest when the initial values are between 15 and 30 are the highest(ranging from 1000 to 3000 variable assignments). The number of variables assigned is between 200 to 500 when the number of nodes are between 1 to 15 and also between 30-40. Whereas the number of variables drastically decrease as the initial numbers assigned become greater than 45.

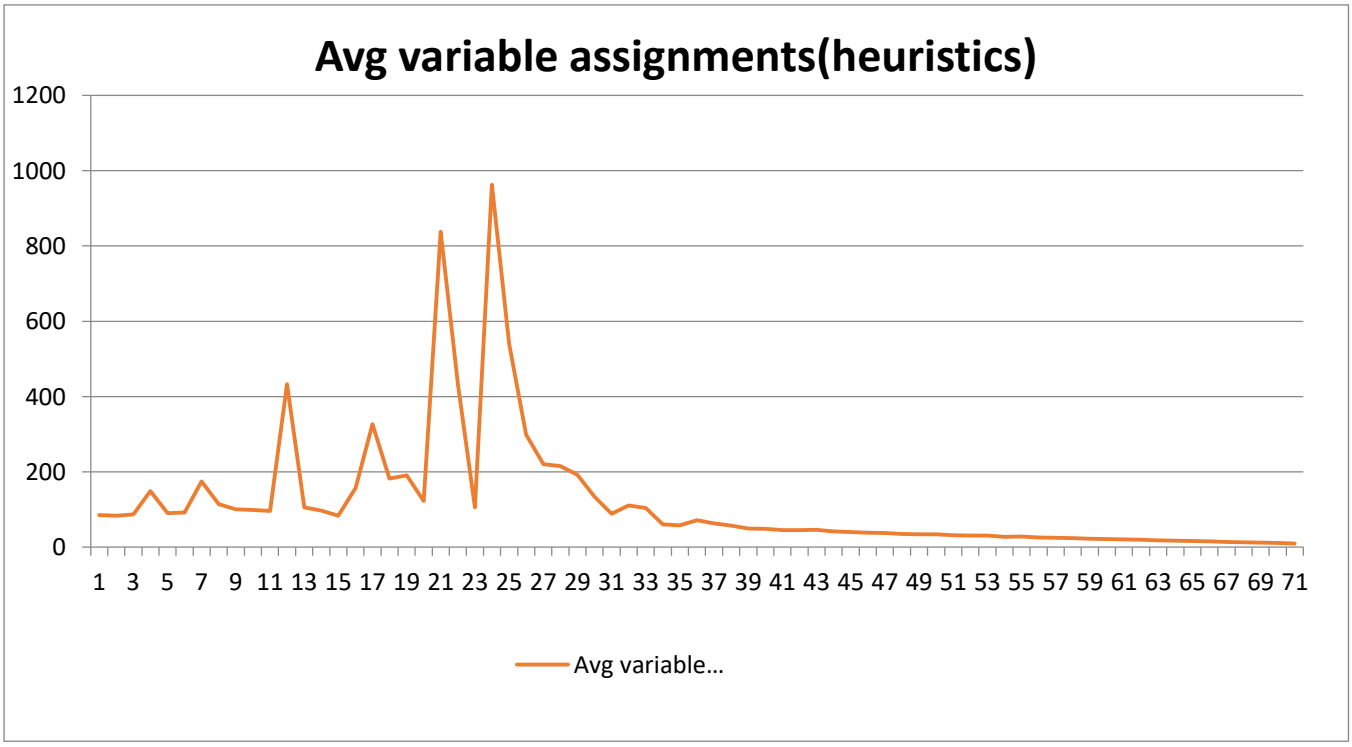
-The same trend is observed when we use forward checking. But the variables assigned while forward checking are comparatively lower than the number of variables assigned while backtracking.

-The solver takes more than 10000 steps in 48 examples out of 700 total examples when it is only backtracking. Whereas the solver takes more than 10000 steps in only 12 examples when it is using backtracking + forward_checking.

- From this we can say that the efficiency and speed of the system increases when it is using forward checking as opposed to using only backtracking.

-This is because, when we use forward checking, the solver checks ahead of time whether an assignment will create possible clashes in the future or not. If it finds that this is the case, then the solver avoids assigning that value to a particular position. So, it prevents certain amount of obvious backtracking. This reduces the number of steps taken by the solver, and thus increases the efficiency of the solver.

Graph for solving Sudoku using backtracking + forward checking + heuristics:



Graph 1: x-axis shows the number of initial values given. Y-axis shows the number of average variable assignments done for each initial value, when the system is using backtracking + forward checking+heuristics

-We can see that the number of variable assignments decreases drastically when using heuristics. This is because heuristics prevent the solver from going into a path whose probability of leading to a solution is very low. So the amount of backtracking is reduced when heuristics are used. So we can say that the solver is most efficient when it uses heuristics.