A) State Space

1.Given the starting state my plan of solving the Sudoku using State Search Representation is as follow:

We know that the sudoku must satisfy below 3 property (for N*N). i.e.

- Each row contains all numbers from 1 to N.
- Each column contains all numbers from 1 to N.
- Each Sub-grid contains all number from 1 to N.

So, problem state space is defined as a follow:

Initial State: The given N*N grid with a partially filed number which satisfy above three property. the instance of problem is as given below as N=9:

		3		2		6		
9			3		5			1
		1	8		6	4		
		8	1		2	9		
7								8
		6	7		8	2		
		2	6		9	5		
8			2		3			9
		5		1		3		

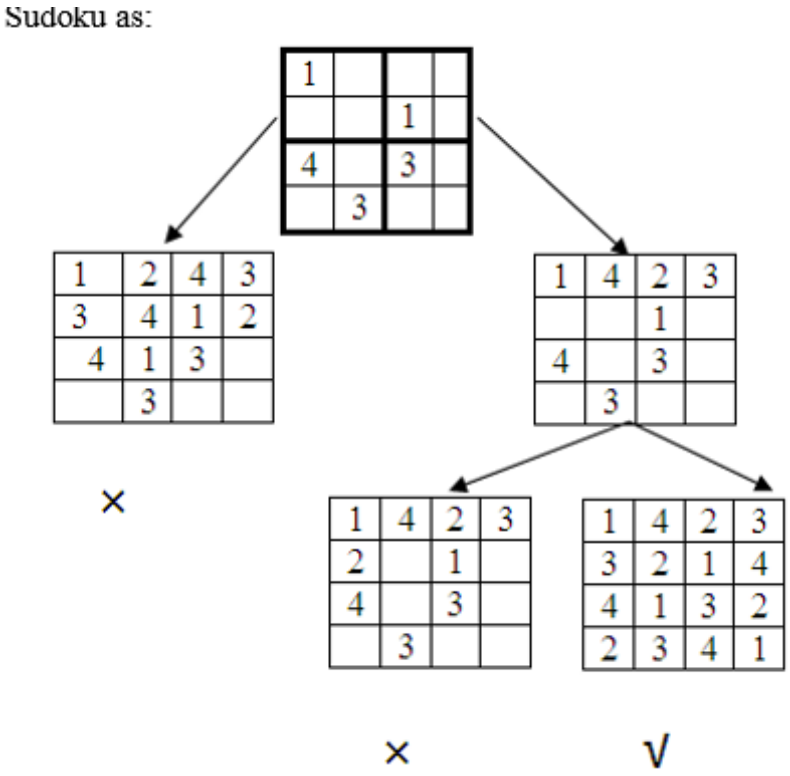
Goal State: We have to fill remaining place with a number between 1-N such that they don't violate above three property. For above instance the goal state is as follow:

4	8	3	9	2	1	6	5	7
9	6	7	3	4	5	8	2	1
2	5	1	8	7	6	4	9	3
5	4	8	1	3	2	9	7	6
7	2	9	5	6	4	1	3	8
1	3	6	7	9	8	2	4	5
3	7	2	6	8	9	5	1	4
8	1	4	2	5	3	7	6	9
6	9	5	4	1	7	3	8	2

Action or Operation: This step defines the operation that can be apply on each state such that we can achieve the goal state. Here from the problem it each clear that at each step we search for empty box in N*N sudoku and then we try all the possible solution 1-N and we apply this step till it satisfy above three property or we reach to goal state.

Possible Solution: (when N=9) 6.67×10^{21} solution is possible.

Each step of Tree (for N=4) which will be generated can be the show below:



And it goes on and on till find a solution.

B) Traversal Time Complexity

Assuming that travers naively state space using the brute force searching algorithm using backtracking we have following complexity in terms of Big-O.

The recursion tree will branch here into 9 (for N=9) branches and at each sub level the problem tree branching size is reduced by one. The problem for a grid size of N*N where N is a perfect square. let M = N*N, the recurrence equation given as

T(M) = 9*T(M-1) + O (1)

where T(N) is the running time for a problem size of N.

Solving this recurrence will yield, O (9^M). Here M=9*9 hence O (9^{81}).

For the N*N grid with P as (1-P) as input we have complexity O (P^{N*N}).

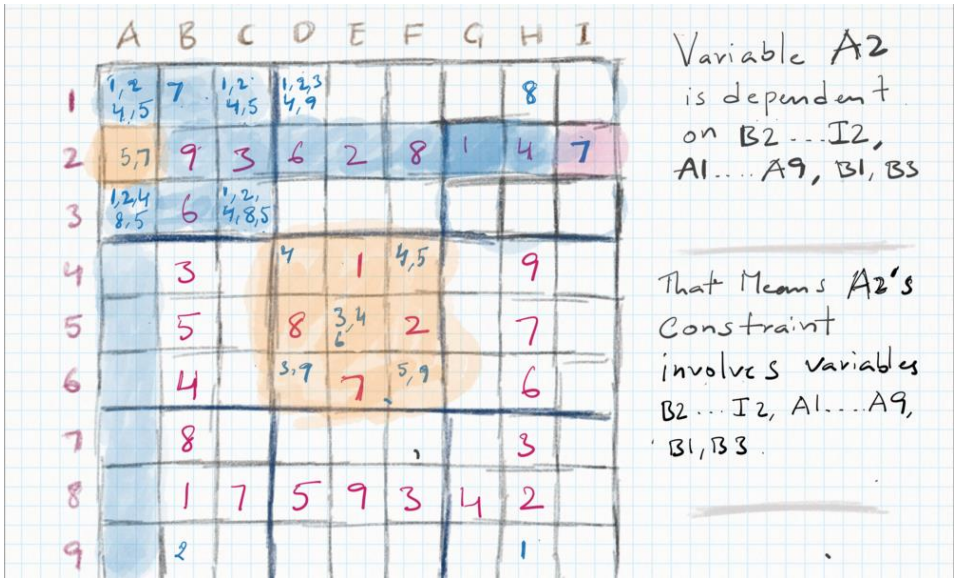
C) Heuristic Search

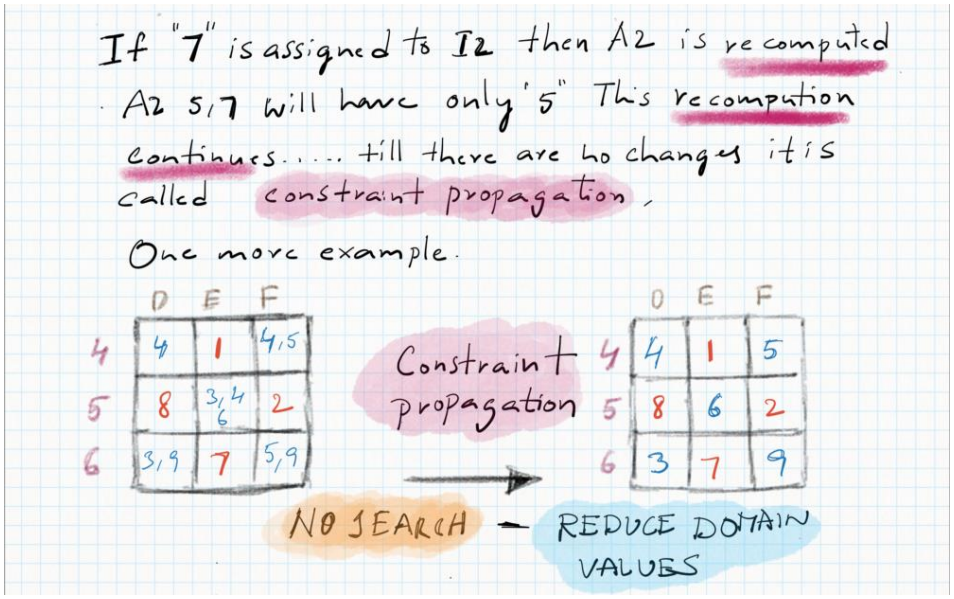
We implement the search problem with Constraint propagation. Here as a cost function we maintain the set for each block in sudoku and update each set via above three constraint. And then we find a least possible set of values for particular position and apply that value to all the remain block till we find solution. Here we reduce the space search by applying implications of the given number present in initial state.

Here we use two strategies.

- If a square has only one possible value, then eliminate that value from a set of rows, column and the sub set.
- If a unit has only one possible place for a value, then put the value two that variable and remove from there row, column and sub-set.

Visualization of it as a tree as N=2 is as a follow:





D) Pseudocodes:

Backtracking:

```
bool Solve(sudoku conf) {
 if (no choices) // BASE CASE
     return (conf is goal state);
   for (all available choices) {
            try one choice x;
          If ( Solve (conf with choice x is made till end)){
                    return true;
                    undo choice x;
    }
    return false;
}
```

```
bool SolveSudoku(Grid<int> &grid)
{
 int row, col;
   if (!GetemptyLocation(grid, row, col))
    return true;
   for (int i = 1; i <= 9; i++) {
  if ( Checkforproperty(grid, row, col, i)) {
       grid (row, col) = num;
   if (SolveSudoku(grid))
        return true;
  grid(row, col) = UNASSIGNED;        // undo & try again
} }
  return false; //  triggers backtracking  }
```

Constraint Propagation:

Repeat following steps until the puzzle is solved:

a) Select a set which contains just one element (if there is none, then try every option in that)

b) Set the corresponding square of grid the appropriate value

c) Propagate implications resulting from this choice to other grid cells

(1) The value just given to the cell can be removed from the sets of possible values for other cells in the same row, column, and block.

(2) The square can be removed from the sets of possible locations for unused values in the row, column, and block of which it is a part