Artificial Intelligence (CSE 537) Homework 2
Name: Rohan Vaish
SBU ID: 111447435

Section 1:

How each algorithm works?

->DFSB:

Depth First Search Backtracking focuses on assigning available color to the state on a particular level based on the already assigned colors in the previous levels. As a part of this step, it also checks for safety of this assignment (compatibility with neighbors), if it returns True, it then passes that assignment downwards to the next level (next state-color assignment using temp_cur_assignment) for further assignments for other states. If on the last level (assignment of last state-color pair), solution is found, it returns that solution. The first instance of a valid assignment as soon as found, is sent back as the assignment. If no assignment is found on the last level, we return None.

Pseudo Code:

```
function dfsb_solution(csp):
  return recursive_dfsb(0,empty_assignment{})

function recursive_dfsb(variable_to_assign, current_assignment):
  if variable_to_assign is Number_of_states:
       return current_assignment
  else:
       state_to_color= graph[variable_to_assign]
       for color in domain of state_to_color:
          if isSafe(current_assignment, state_to_color, color):
             temp_cur_assignment= backup(cur_assignment)
             temp_cur_assignment[state_to_color]=color
             temp_assignment= recursive_dfsb(next_variable,temp_cur_assignment)
             if temp_assignment not None:
                return temp_assignment
       return None
```

->DFSB++:

Depth First Search Backtracking++ focuses on assigning least constraining available color to the most constrained state on a particular level based on the already assigned colors in the previous levels. As a part of this step, it also checks for Arc-Consistency and prunes the domains of each state according to current assignment. If, Arc-Consistency returns False (domain of one of the unassigned states becomes Null), we revert to the initial state of the group before last assignment.

It, then passes that assignment downwards to the next level (next state-color assignment using temp_cur_assignment) for further assignments for other states. If on the last level (assignment of last state-color pair), solution is found, it returns that solution. The first instance of a valid assignment as soon as found, is sent back as the assignment. If no assignment is found on the last level, we return None.

Pseudo Code:

```
function dfsb_plus_solution(csp):
  return recursive_dfsb(0,empty_assignment{})

function recursive_dfsb_plus(variable_to_assign, current_assignment):
  if variable_to_assign is Number_of_states:
       return current_assignment
  else:
       state_to_color= most_constrained_state(current_assignment)
       color_order= least_constraining_colors_order(state_to_color, current_assignment)
       for color in color_order:
          backup_graph=back(graph)
          graph[state_to_color]->color=color
          if AC3(current_assignment, state_to_color):
             temp_cur_assignment= backup(cur_assignment)
             temp_cur_assignment[state_to_color]=color
             temp_assignment= recursive_dfsb_plus(next_state,temp_cur_assignment)
             if temp_assignment not None:
                return temp_assignment
          graph=backup
       return None
```

->Minimum Conflicts:

A random assignment of colors to states is selected, and then the algorithm randomly selects a state from the set of conflicting variables (according to constraints). Then we assign a color to this variable which minimizes the conflicts, if there are more than one such colors, select one randomly. The process is iterated until a solution is found (with 0 conflicts) or until the max iterations have been reached. If no solution is found, we start the whole process with a different random initial assignment until a solution is found or the maximum number of random restarts is reached. This is in a way a local search algorithm, as it works on reducing the number of conflicts by changing the color of one of the conflicting states.

Pseudo Code:

```
function minconflicts_solution(csp):
  for random_restarts in MAX_RESTARTS:
       cur_assignment=random_state()
       for trial in MAX_TRIALS:
              conflicts=0
              conflicts_states=[]
              update_conflicts(cur_assignment)
              if conflicts=0:
                assignment=cur_assignment
                return assignment
              else:
                next_state=random_state_select(conflicts_states)
                next_state_color=min_conflicting_color(next_state,cur_assignment)
                cur_assignment[next_state]= next_state_color
  return None
```

Section 2:

Table describing the performance of each algorithm:

Input File	Parameter	DFSB	DFSB++	MinConflicts
backrack_easy	Time Taken(ms)	0.408	1.446	0.261
	Steps Taken	8	16	15
backrack_hard	Time Taken(ms)	No answer (Time out)	6167.303	18457.568
	Steps Taken	No answer (Time out)	1003	27816
minconflicts_easy	Time Taken(ms)	45.472	10.915	0.720
	Steps Taken	1505	50	30
minconflicts_hard	Time Taken(ms)	No answer (Time out)	327.085	247.824
	Steps Taken	No answer (Time out)	240	2047

Section 3:

Observed performance differences:

->DFSB:

DFSB or plain Depth First Search with Backtracking cannot solve most problems in reasonable amount of time. If it finds solution, it explores a lot more states than DFSB++ or Minconflicts algorithms. This is because of not forward checking at each step or pruning the arcs at each step to deduce failure earlier. Another reason for it to be time taking is that is assigns colors to states in uninformed manner without keeping most constrained states or least constraining color in mind.

For example:

States explored for minconflicts_easy= 1505 >> 50 or 30 by DFSB++ or Minconflicts respectively

->DFSB++:

DFSB++ is an improvement over DFSB algorithm. It solves all the problems that DFSB cannot. By selecting most constrained states and least constraining color for assignment and pruning the arcs at every steps, it can deduce failure over that branch of tree earlier hence will take less time and steps. The improvement can be seen by it's ability to solve the problems DFSB couldn't and solve the ones which DFSB could but in lesser steps (50 for minconflicts_easy while DFSB takes 1505)

->Minconflicts:

Minconflicts is a type of local search algorithm. The only problem is to avoid local maxima and plateaus which it deals with by random restarts after a certain number of conflicts reduction trials. This algorithms find solution for all the cases though with different running times and number of steps due to randomization. One observation was it takes more steps to reach a solution than DFSB++ for harder problems. If solutions are uniformly distributed, minconflicts can find the solution faster due to uniform random restarts.