Gabriello Lima, 112803276

1. Briefly explain how each method works including pseudo code for DFSB, DFSB++, and MinConflicts

- **DFSB**- Iterate through assigning colors as we go. If an assignment breaks the constraints, backtrack up a level and assign it a different color.

  Pseudocode:

  ```
  DFSB(assignment)
      If assignment is solution
          Return assignment
      For each variable we can assign a color:
          For each color we can assign to this variable:
              Solution = DFSB(assign variable color, pass new assignment)
              If solution is not failure
                  Return solution
      Return failure
  ```

- **DFSB++**- Smartly iterate through, assigning colors as we go. We do this by using the most constrained variable (i.e. least colors remaining) that isn't assigned, and assigning colors in the order of the least constraining value (i.e. which color prunes the domain of adjacent variables the LEAST). Use AC3 to smartly keep track of edges/constraints/assignments. If an assignment breaks the constraints, backtrack up a level and assign it a different color.

  Pseudocode:

```
DFSB++(assignment)
If assignment causes violation
    Return failure
If assignment is solution
    Return assignment
For each variable in mostConstrainedListOfVariables(assignment):
    For each child in generateSortedChildren(assignment, variable)
        Solution = DFSB++(child)
        If solution is not failure
            Return solution
Return failure
```

- **Min Conflicts**- Assign each variable a random color. If there are any constraint violations, change the color of that variable while trying to minimize the total number of violated constraints. If you get stuck in a local minima, break out and start again.

  Pseudocode:

```
Minconflicts(assignment)
    Assign each variable a random color
    While assignment is not solution
        Var = Pick a random variable that has a constraint violated(assignment)
        Color = Pick a random color minimizing violated constraints (var)
        assignment [var] = color
        If stuck in a local minima
```

Return failure (indicating we start again)

2. Tables describing the performance of the algorithms (DFSB, DFSB++, and MinConflicts) on your generated problems. Please refer to section 3.5.

DFSB

Parameter	Number of states	Time (ms)
20, 50, 2	30 +- 0	0.9 +- 0.30
50, 100, 3	N/A	N/A
100, 750, 3	N/A	N/A
200, 10000, 4	N/A	N/A
400, 40000, 4	N/A	N/A

DFSB ++

Parameter	Number of states	Time (ms)
20, 50, 2	20 +- 0.0	6.8 +- 4.07
50, 100, 3	50 +- 0.0	31.65 +- 1.871
100, 750, 3	100 +- 0.0	139.15 +- 8.06
200, 10000, 4	200 +- 0.0	1156.75 +- 11.63
400, 40000, 4	400 +- 0.0	8691.3 +- 33.187

Min-conflicts

Parameter	Number of states	Time (ms)
20, 50, 2	41.4 +- 16.59	2.55 +- 0.944
50, 100, 3	28903.2 +- 33391.53	3570.1 +- 4115.48
100, 750, 3	410 +- 127.1	311.9 +- 94.68
200, 10000, 4	733.9 +- 91.72	8767.05 +- 1110.61
400, 40000, 4	NA	NA

3. Explain the observed performance differences

DFSB is clearly the worst here because we're blindly searching. Although the time spent per state is low, we are basically brute forcing it. For DFSB++, it's closer to an informed algorithm. We're spending more time per state to prune good assignments. Hence DFSB++ is able to handle more complicated problems. For Min-conflicts, it's similar to DFSB++ in the sense that we're pruning our domain to get good assignments, but there's a little bit of randomness involved as it's a local search. Hence, if it accidentally goes down the wrong path, it could end up taking an extremely long amount of time compared to another algorithm, such as DFSB++ (see parameter

#2 for Min-conflicts above). However, if it guesses correctly, it could take less time compared to another algorithm, such as DFSB++ (see parameter #3 for Min-conflicts above)