Assignment 2

Name: Sandeep Kumta Vishnu

SBU_ID: 111482809

1. DFSB, DFSB++ and Min Conflicts

1.1 DFSB

This search algorithm is an exhaustive search method which basically expands up all the nodes in a given path without any pruning mechanism. This algorithm is efficient only when the search space is i.e number of nodes in the graph is significantly low. It worked properly for both backtrack_easy and backtrack_hard input files but took almost a minute to solve the backtrack_hard input.

Pseudocode

```
#input: inital assignment
#output: solution assignment or 'failure'
function dfsb(assignment):
    if elapsed_time > 1 minute:
        return 'failure'
    if all the positions are colored completely:
        return assignment_list
    unassigned_position = find first unassigned position in the assignment_list
    for each color in colors:
        assignment_list[unassigned_position] = color
        if assignment is consistent:
            result = dfsb(assignment) #current_assignment
            if result is not failure
                return result
        remove the above assignment
    return 'failure'
```

1.2 DFSB++

This search algorithm makes use of below 3 heuristic algorithms to perform pruning, to reduce branching factor and to reduce the search space. Its pseudocode is explained after explaining each of the heuristic algorithms.

1.2.1 AC-3 Algorithm (Arc Consistency Algorithm) AC-3 algorithm is actually a variant of forward checking algorithm. This heuristic is used to examine how the domain of neighbors and the nodes connected to those neighbors will be affected if a particular color is assigned to a node. Arc from X->Y is consistent only when for every value x of in domain of X there is some allowed y from domain of Y. This algorithm removes the inconsistent values from domain of all arcs. If domain X loses a value, neighbors of domain X need to be rechecked because its incoming arcs may become inconsistent.

Pseudocode

```
function ac3():
    while arcs is not empty:
        arc = remove first element from arcs list
        # arc[0] arc[1] means x & y where there is a direct edge (x,y)
        check if there is inconsistency in the domains of arc[0] and arc[1]
        if there is inconsitency:
            remove inconsistent values from domain of arc[0]
            increment number of prunes
            removed flag = True
        if removed is True:
            for each neighbor in constaints[arc[0]]
                if neighbor is not arc[1]
                    then add new arc from neightbor to arc[0]
```

1.2.2 Minimum Remaining Values(MRV) This heuristic is used to find the position of the state where we want to place next assignment. It chooses a state with least number of legal values left. By this, we get a position which will fail first if that position is not chosen next. This greatly helps in reducing the branching factor of the dfsb search.

Pseudocode

```
#output: position with least legal values remaining
function minimum_remaining_value():
    current_minimum_remaining_value = total number of colors
    unassigned_position = 0
    for each position in assignment_list:
        if position is not assigned:
            legal_values_left = total number of colors - find number of illegal values at po
            if legal_values_left <= current_minimum_remaining_value:
                current_minimum_remaining_value = legal_values_left
```

```
                unassigned_position = position
    return unassigned position
```

1.2.3 Least Constraining Value(LCV) This algorithm takes a state to be chosen next, and it chooses the values of colors that this variable can take which will affect its neighboring nodes. This improves the dfsb++ algorithm because the value which affects its neighbor the least is very less likely to cause failures.

Pseudocode

```
# constraint_list is adjacency list
function least_constraint_value(position):
    colors = initialize list of size number of colors to 0
    if constraints_list[position] has no neighbors:
        return list in range 0 to total number of colors
    illegal_values_set = empty_set()
    for each color from 0 to total number of colors:
        illegal_values_set.add(color)
        for each neighbor of position:
            if neighbor is not assigned and neighbor has other neighbors:
                # next level neighbor of 'position'
                for each next_neighbor in neighbor of position:
                    if next_neighbor is assigned:
                        illegal_values_set.add(assignment of next_neighbor)
                colors[color] += total number of colors - total number of illigal values
                remove all other elements from the set except the current color
    return the list of positions in the descending order of values in colors list
```

Pseudocode for DFSB++

```
#input: inital assignment
#output: solution assignment or 'failure'
function dfsb_plus_plus(assignment):
    if all the positions are colored completely:
        return assignment_list
    #For forward checking
    call arc_consistency algorithm for forward checking

    # to choose state with least legal values left
    unassigned_position = call minimum_remaining_values heuristic function

    # to get list of colors to assign to the above position
    # which affects the neighboring states the least
    call least_constraining_heuristic(unassigned_position)
```

```
    for each color in colors:
        assignment_list[unassigned_position] = color
        if assignment is consistent:
            result = dfsb_plus_plus(assignment) #current_assignment
            if result is not failure
                return result
        remove the above assignment
    return 'failure'
```

1.3 Min Conflicts Algorithm

This algorithm is a randomized local search algorithm. Initially it greedily assigns colors to each states in such a way as to minimize the number of conflicts. The till a consistent solution is available, among the list of conflicting states it choses a conflicting state randomly then for that state it chooses a color in such a way as to minimize the number of conflicts. If multiple possibilities exist, it chooses a color randomly. After certain number of steps, random restart strategy is used to avoid the states getting stuck in local minimum.

Pseudocode

```
#input: maximum iteration
#output: boolean whether success or not
#assignment_list is global
function minimum_conflicts(max_iteration):
    greedily assign each state a color in a way to minimize the conflicts
    for each iterations till max_iteration:
        if assignments are consistent:
            return True
        choose a random state where there is a conflict
        assign a color to that state in a way to minimize the conflicts
        # if multiple possibilities exists, then assign random possibility
        increment the iteration
```

2. Performance Table

Sl. No.	Algorithm	Input File	Time (ms)	# of Searches	# of Arc Prunings
1	DFSB	backtrack_easy	0.0	8	0
2	DFSB	backtrack_hard	59968.75	54926	0
3	DFSB++	backtrack_easy	0.0	8	10
4	DFSB++	backtrack_hard	12515.625	2679	1389
5	Min Conflicts	minconflict_easy	15.625	133	N/A
6	Min Conflicts	minconflict_hard	No Answer	12000	N/A

3. Observed Performance difference

1. DFSB is just a naive backtracking algorithm. As the number of nodes increases the space state graph increases and DFSB becomes inefficient.
2. Improved DFSB is very efficient as it prunes the inconsistent states using AC3 heuristic, chooses the effective position to place the next color using MRV heuristic and choose best list colors to place next as to minimize the effect on neighbors using LCV heuristic.
3. Minimum conflicts is a randomized algorithm which tries to minimize the conflicts in each step. The implementation here uses random restart technique to overcome plateaus and local minimums. Performance is improved from random restart technicque..

References:

1. Class Slides
2. Introduction to Artificial Intelligence: A Modern Approach (Text Book)
3. https://people.cs.pitt.edu/~wiebe/courses/CS2710/lectures/constraintSat.example.txt
4. http://www.cs.cornell.edu/courses/cs4700/2011fa/lectures/05_CSP.pdf
5. http://pami.uwaterloo.ca/~basir/ECE457/week5.pdf