Graph Coloring using CSP and Min-conflicts Local Search

Assignment 2

Name: Harsha Chandwani

SBU ID: 111481387

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

Solution:

1.DFSB

Pseudocode:

Algorithm DFSB(A,CSP)
If A is complete
    Return A
endif
Remove a variable V from CSP.Unassigned
For all values v that belong to the D(V) do
        If v is consistent with A according to the constraints then
                Add V = v to A
                Result = DFSB(A,CSP)
                If result != failure then
                        return result
                endif
                remove V = v from A
        endif
End for
Return failure

DFSB Explained:

The DFSB starts with an empty assignment and a list of unassigned variables in the CSP. The output is either failure or a complete assignment. We start with assigning a value to a variable, checking if it is consistent with the neighbours, if yes, then move on to the next variable assignment. If the value is found as inconsistent at a particular stage, we backtrack and continue with the next value for a variable.

2. DFSB++

Pseudocode:

```
Algorithm DFSB++(A,CSP)
If A is complete
    Return A
endif
Select the Most Contrained Variable V from CSP.Unassigned
current_domains_copy = CSP.current_domains
D(V) = Order the current Domain Values of V in the least constraining order
For all values v that belong to the D(V) do
        If v is consistent with A according to the constraints then
                Update CSP.current_domain[V] = [v]
                Add V = v to A
                queue = (head,tail) for head in CSP.neighbours[V]
                res = AC3(CSP, queue)
                If res != failure
                        res = DFSB++(A,CSP)
                        If res != failure then
                                return result
                endif
        remove V = v from A
        CSP.current_domains = current_domains_copy
        endif
End for
Return failure
```

DFSB++ Explained:

This algorithm is better than plain DSFB in terms that it chooses the most constrained variable meaning the variable that has the least number of options for coloring available. Also, it assigns this variable with the least costraining value meaning a value that causes minimum effect on the neighbor assignments. This is definitely more effective than naïve DFSB. It performs an ARC consistency check once a variable is assigned, to make sure that this current assignment would not lead to an inconsistent state in future. If it does, it eliminates values from the domain of the variable that make the arc inconsistent and insert to the queue arcs that are affected by that change in domain values.

3. Minconflict

Pseudocode

```
Algorithm MinConflicts(A,CSP):
A = Random assignment for the variables in the CSP
While restart < 100:
        IsSolution(A)
                Return A
        If CSP.attempts > CSP.limit
                A = Random assignment for the variables in the CSP
                CSP.attempts = 0
        CSP.attempts = CSP.attempts + 1
        Var = getRandomConflictingVariable(A,CSP)
        Value = the value v for var that minimizes the conflicts i.e Least Constraining value
        set Var=Value in A
Return Failure
```

MinConflicts explained:

The algorithm starts with an initial random assignment to the variables. The algorithm then randomly selects a variable form the conflicting ones. Then it assigns this variable the value that results in the least number of conflicts. If there are more than such variables, it selects one of those, randomly. This process of randomly selecting a variable and assigning it a minimum conflicting value goes on until a solution is found or a limit is reached.

2. A table describing the performance of the algorithms (DFSB, DFSB++, and MinConflicts) on the sample problems. Use the actual time taken, and the number of search + arc pruning calls in DFSB and number of search steps in MinConflicts to compare.

Input file	Algorithm	Time Taken(ms)	Steps
backtrack_easy	DFSB	0.0283	9
	DFSB++	0.3640	17
	Min Conflicts	1.4448	9
Backtrack_hard	DFSB	Time out	NA
	DFSB++	860.9294	1002
	Min Conflicts	152498.6018	22099
Minconflict_easy	DFSB	5.2231	1506
	DFSB++	2.4987	51
	Min Conflicts	23.6821	117
Minconflict_hard	DFSB	Time out	NA
	DFSB++	52.1333	241
	Min Conflicts	7556.2866	4890

Performance Differences Explained:

1.DFSB

This is inefficient when compared it the DFSB++ as it explores a lot of states. If we compare the number of states in the case of minconflicts_easy file, DSFB explores 1506 states whereas DFSB++ explores just 51 states.

2. DFSB++

This is efficient when compared to the DFSB. It is basically DFSB with intelligence. It does involve a bit of overhead in the sense it does the arc pruning every time an assignment is done. But the number of states explored are significantly lesser than DFSB. All the instances that were not solvable by DFSB are definitely solvable by DFSB++. The reason behind the superior performance of DFSB++ is that it uses the computation time to compute the heurisitics even in the case of simple inputs.

3. Min conflicts.

The performance of this algorithm highly relies on the initial random assignment. Due to high level of randomness involved, it is difficult to say we find a pattern in the running time of the algorithm. Sometimes the search gets stuck in the pleateau and for this reason, I have included random restart wherein a we start afresh from a new random state to avoid getting stuck in pitfalls of local search.