**Assignment 2 -Report**

SBU ID:110949097

Bharatkumar Darapu

Input :backtrack\_1.txt,backtrack\_2,minconflict\_1,minconflict\_2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Search Algorithms** | **Time Taken(in seconds)** | **No. of nodes Explored** | **Arc-Pruning calls** | **No. of Search steps** |
| **DFSB** | Running for longer than a minute for backtrack\_1 and backtrack\_2, 0.004513,0.001502 | 89,100,12,12 | NA |  |
| **DFSB++** | 0.288153,0.627734,0.006517,0.002005 | 200,120,12,12 | 299,717,34,21 |  |
| **Min-Conflicts** | 2.060 ,3.9765, 0.002004,0.003011 | 120, 200, 12 , 12 | NA | 2981,3411,32,44 |
| **Min-conflicts** | 1.65122,9.0069,0.009055,0.007018 | 120,200,12,12 | NA | 1711,7734,180,130 |

Min-conflicts values noted for two instances .

Running the programs:

**for dfsb.py**

Py dfsb.py <mode-1(dfsb) or 2(dfsb++)> <INPUT file path> <OUTPUT file path>

**For minconflicts.py**

Py minconflicts.py <INPUT file path> <OUTPUT file path>

Output files : dfsb\_minconflict\_1 , dfsb++\_backtrack\_1, minconflicts\_backtrack\_1……………

dfsb\_minconflict\_1 is the output for input minconflict\_1 with dfsb

dfsb++\_backtrack\_1 is the output for input backtrack\_1 with dfsb++

minconflicts\_backtrack\_1 is the output for input backtrack\_1 with minconflicts ……..

Performance Differences:

**DFSB :**

DFSB chooses an unassigned variable and tries all values in the domain of that variable in turn to find the solution, if an inconsistency is detected, then the method BACKTRACK returns failure, then the previous recursion is called to check for the next possible value and this process repeats until a solution is found.

Running time complexity is O(d^n) where d is the number of domain values and n is the number of variables.

For a problem with less number of variables checking all these states might be fast but for the problems with more variables like in backtrack\_1 or backtrack\_2 it will take a long time to check all the possible states.

So in the case of backtrack\_1 and backtrack\_2 this runs for a long time to check through all possible paths until a solution is found.

Minconflicts and dfsb take almost the same time for the given small inputs.but minconflicts can find a solution for backtrack\_1 and backtrack\_2 in less time than dfsb. The minimum conflicts heuristic helps minconflicts to achieve the solution. Minconflict selects a variable from the conflicted variables set and tries to overcome that conflict at each step where as dfsb simply tries for all possible values.

**DFSB++ :**

DFSB++ has three major improvements over dfsb.

First the unassigned variable chosen is the variable with minimum number of domain values remaining.By picking such a variable the search tree can be pruned faster- likely to cause a failure earlier.

Second , value selection for the unassigned variable is not any value – it chooses the least constraining value trying to leave more choices for the next variable assignments.

Third , after each assignment arc consistency is checked for the neighbors and neighbors of neighbors.

After the assignment of a value to the variable ,ac3 performs arc consistency check .In this process if the domain of any variable is reduced to an empty set , we can immediately backtrack from that point without going through the extra effort of expanding the tree from there.

These 3 factors contibrute to the effectiveness of DFSB++.With these three heuristics DFSB++ can break conflicts and find solutions to all the inputs.

For easy inputs(minconflict\_1,minconflict\_2) dfsb runs faster than DFSB++ beacuase it runs without going through these 3 extra heuristics used by DFSB++.

DFSB++ achieves the solution faster than minconflict for the inputs backtrack\_1 and backtrack\_2 due to the 3 deterministic heuristics in dfsb++ , where as minconflicts randomly selects a conflicted variable and tries to overcome that conflict in the next assignment.

**Min\_conflict local search :**

In this method initially ,we choose a random assignment to each variable and try to reduce the number of conflicts between the variables. A random selection is made from the conflicted set of variables , a value is chosen for that variable in such a way that it tries to reduce the number of conflicts with that assignment. If at any instant , if the number of conflicts is reduced to zero , then a solution has been found. Local searches face the problem of running into plateaux , there may be situations when the number of conflicts in an assignment has been reduced to one or two but no further than that. Allowing side way moves to states with same score might help local search to get out of this plateau.

Max steps used for this 10000.

(for backtrack\_2 and backtrack\_1 the max\_steps has to be increased to solve the problem at **all instances** , with 10000 steps most of the times a solution is obtained.)

I resolved the plateau in case of minconflicts by keeping a count for the number of conflicted variables in each assignment, if this values remains constant over a continuous set of assignments , then I am selecting a random value for the variable instead of least constraining value.

REFERENCES: Artificial Intelligence: A Modern Approach by Stuart Russell and Peter Norvig