Project Report

Project on Tree Search Library.

CS2:102 – Data Structures and Algorithms in C

Professors R. Kiran and S. Gujar

Spring 2021, Semester 2 (of 3), Year 1, 2020-21.

IIIT Hyderabad

Group no.: - 36

Abstract

The following Project report is written for the Tree search Library created by Team 36 under T.A. Arpan Dasgupta, in the Data Structures and Algorithms Course in Year 1, term 2(of3) , Academic year 2020 – 2021. In this, the library was set up and the related questions as asked in the Project manual were answered. The report is typed digitally using Microsoft Word.

Table of Contents

Objectives

Make a general tree search library to Do a Tree Search and Visualise it.

The Project Requirements

The tree node should be generic enough to allow arbitrarily complex information to be stored (say game states and other variables). By default, you might want to store information like time seen, depth etc. to perform different searches.

2. The next node to explore in a tree search can be picked via a generic priority queue instead of a specific data structure like stack or queue in DFS or BFS. The comparator function for the priority queue should be something which can be plugged in by the user to allow them to change the algorithm.

3. Default algorithms - DFS, BFS, and greedy search need to be provided as sample (basically, write the comparator function with required node structure for these algorithms). [Big bonus if you can do MCTS :P]

4. Some information needs to be placed in a global array at each iteration for analytics. Some of this information will be - max depth, average depth, branching factor. You can add more of this if you want. This information can be plotted/analyzed for different algorithms to show the difference. (DFS max depth increases fast, branching factor less, etc.) [Plotting not necessary but would be nice]

Additional Information:

1. The file the user needs to change must be a single file which has the node struct, the comparator for the priority queue and any other necessary function which the user needs to modify.

2. The tree structure will be input beforehand from the user (including required details for each node).

3. The data to test your algorithm on can be picked from somewhere or can be generated or we might provide it to you.

Theoretical: Time and Space Complexity Analysis

Individual Functions : Time Complexity.

Within → Main.c

1. Int main()
    - Not considering the member function functionalities, and if we let Number of Nodes in the testcase to be ‘N’, Input takes O(N) time, and O(N) space.
    - The Initialize tree function is called once, which itself is a O(N* N)time, O(N) space function.
    - Initialize heap function is called once, which itself is a O() function.
    - We call Heap Insert , which itself is a O() function.
    - We take O(k)time to take in input of search mode.
    - We now have an implementation of minheap extract min , which itself is a O() function. Which progresses N times, inserting each element into heap and popping the minheap element and adding its children to the heap. We print the entire input in the order we would traverse in a search to get a particular element.

Within → treesl.c

1. Bool Comparator

   a. For BFS, DFS and Greedy search, It is O(1) time and has a space complexity of O(0)

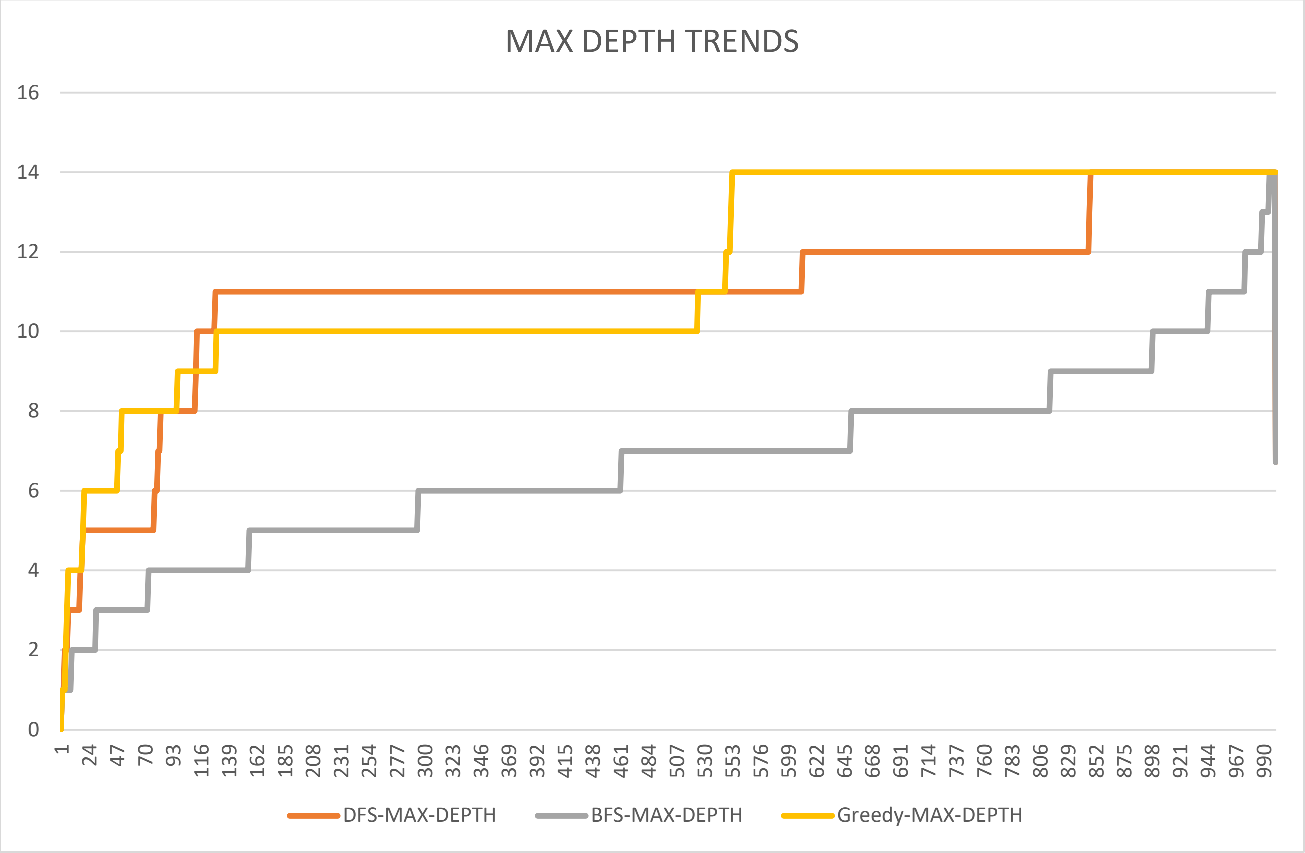
2. Void swap
   a. O(1) time , O(0) space
3. Ptr_Node assign_Node
   a. O(1)time, O(1) space
4. Void initialise_tree
   a. O(N* (2*N))time, O(N) space, as it calls (InsertNODE , N times)
5. Void insertNode
   a. O(2*N), owing to how the Tree is structured. 10000 is the MAX amount of children to each node., if
   b. Takes a space of O(0)
6. Void Printtree
   a. Takes O(N) time and O(0) space
7. PtrHeap Initialise heap
   a. Takes O(1) time and O(N) space
8. Void HeapInsert
   a. Takes O(1 * O(log(2){N}))time , as it calls Percolate Up
   b. Also takes O(log(2){N}) space.
9. Void percolateup
   a. Takes O(log(2){P}) time, when P is the number of elements in the heap. Worst case scenario gives us O(log(2){N})
   b. Takes O(log(2){N}) space. Recursive callson callstack
10. Int deletetop
    a. Takes O(1* O(log(2){N})) time as It calls PercolateDown.
    b. Also takes O(log(2){N}) space
11. Void percolateDown
    a. Takes O(log(2){P}) time, when P is the number of elements in the heap. Worst case scenario gives us O(log(2){N})
    b. Also takes O(log(2){N}) space.

12. Int Isempty
    a. Takes O(1) time and O(0) space.
13. Void Print
    a. Takes O(N) time and O(0) Space

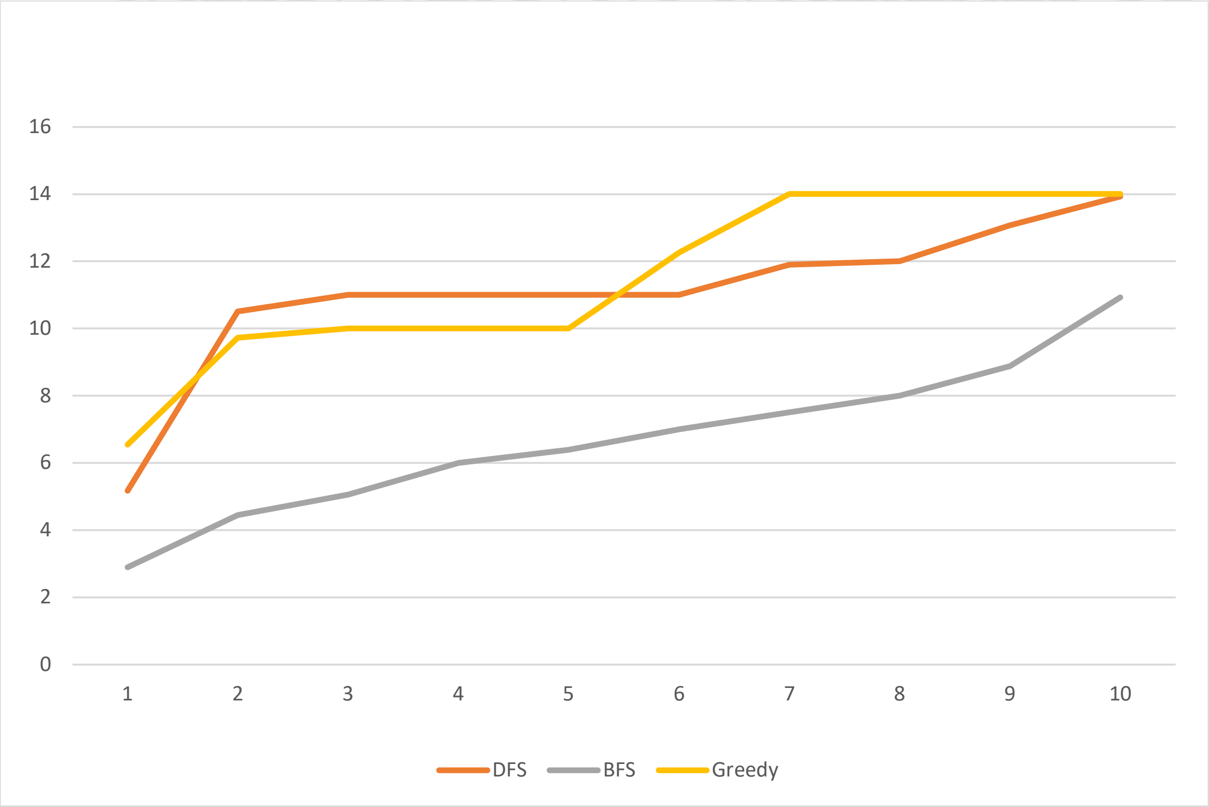
Practicals : Data Analytics. On Testcases.

We have plotted the graphs for the outputs for Testcases 3 In BFS, DFS and Greedy search

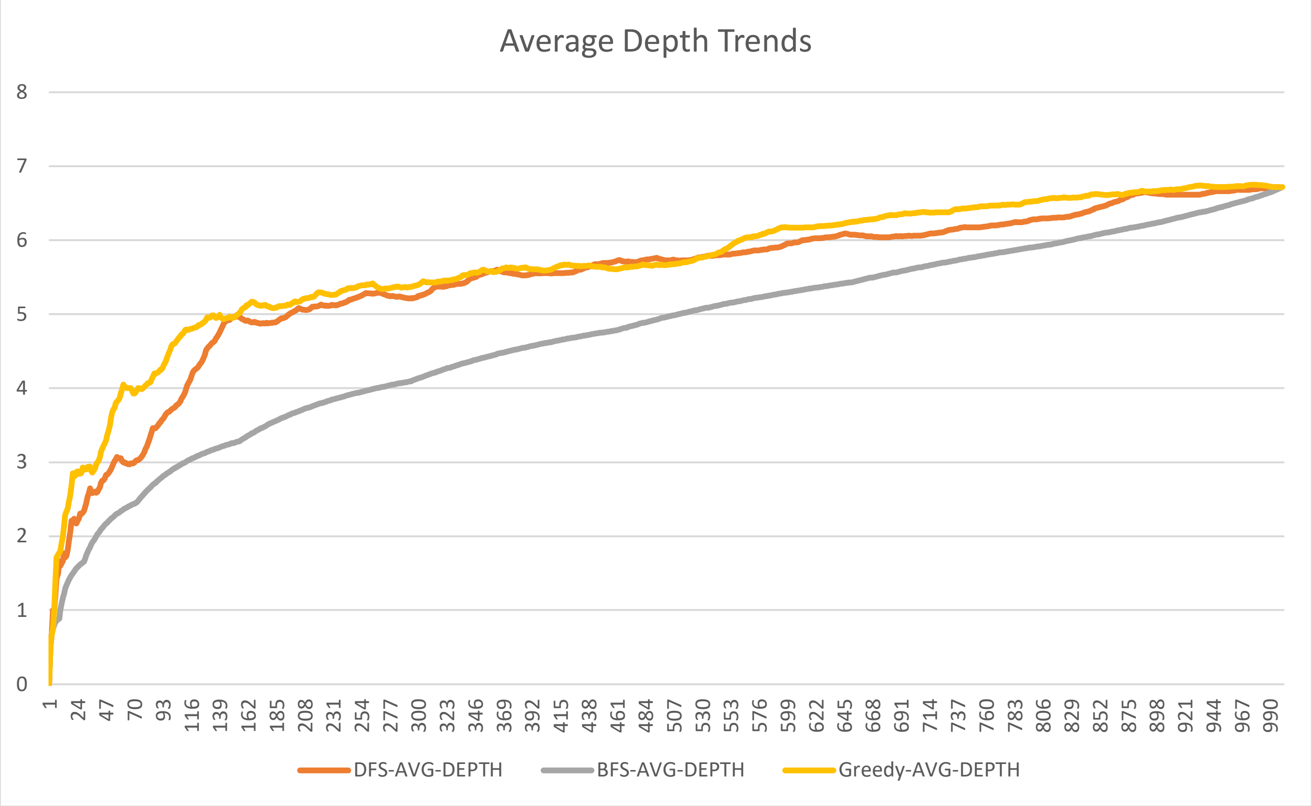
Max Depth Trends →

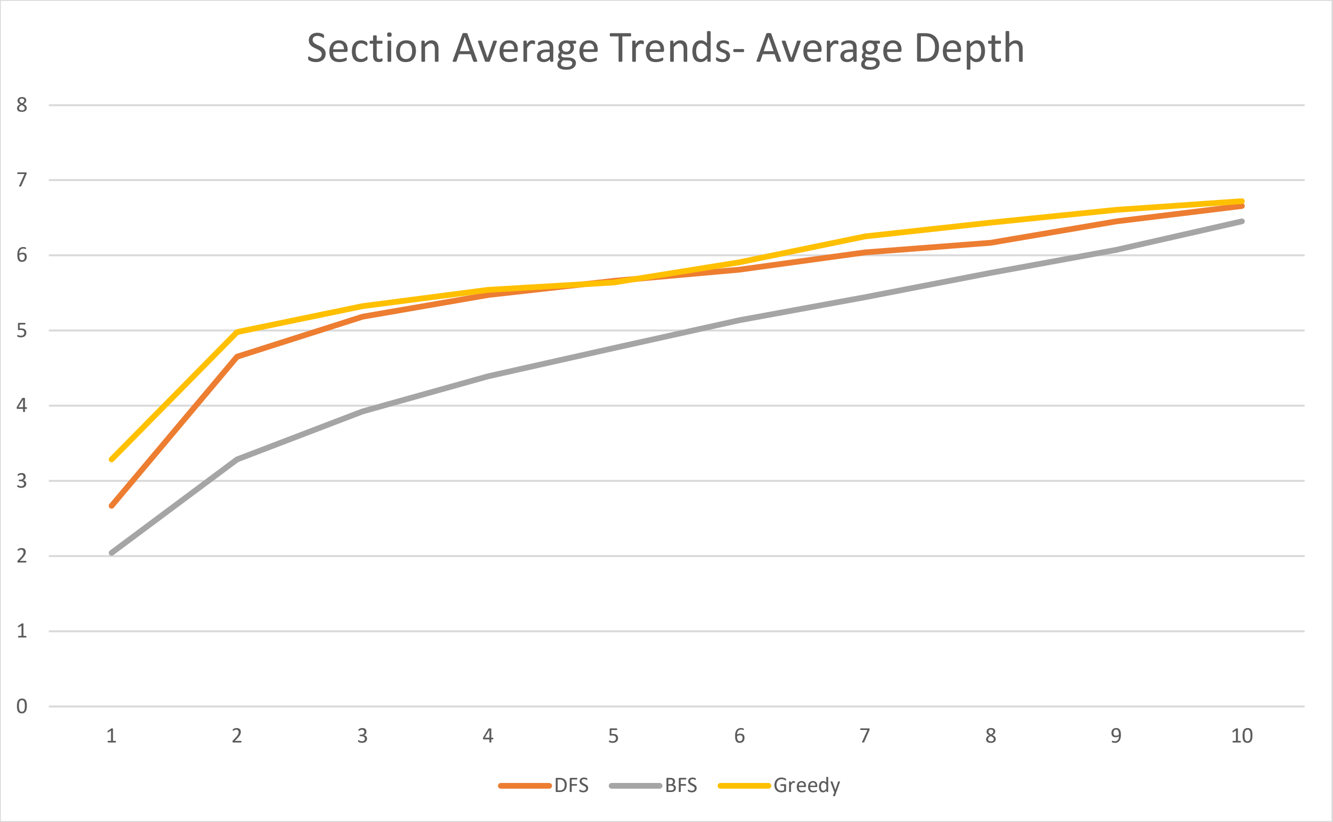


Sectional Average Trends

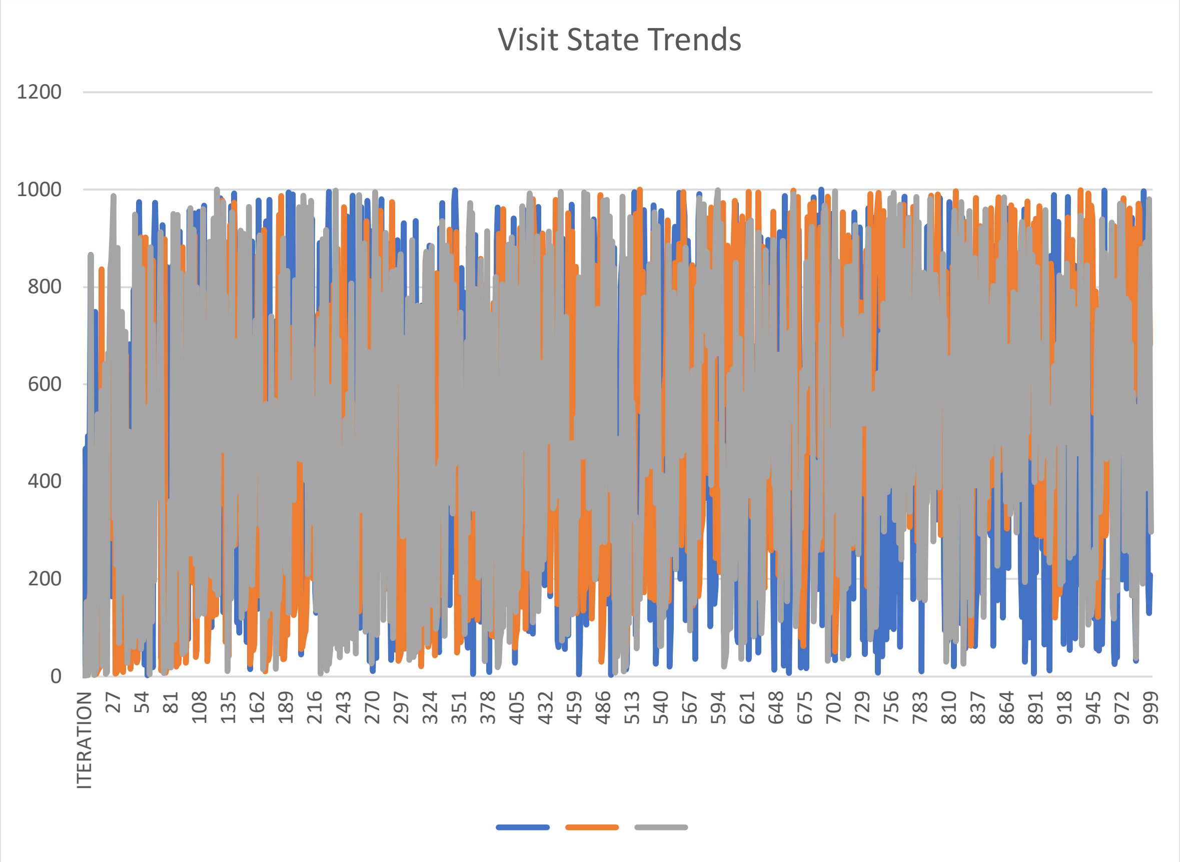


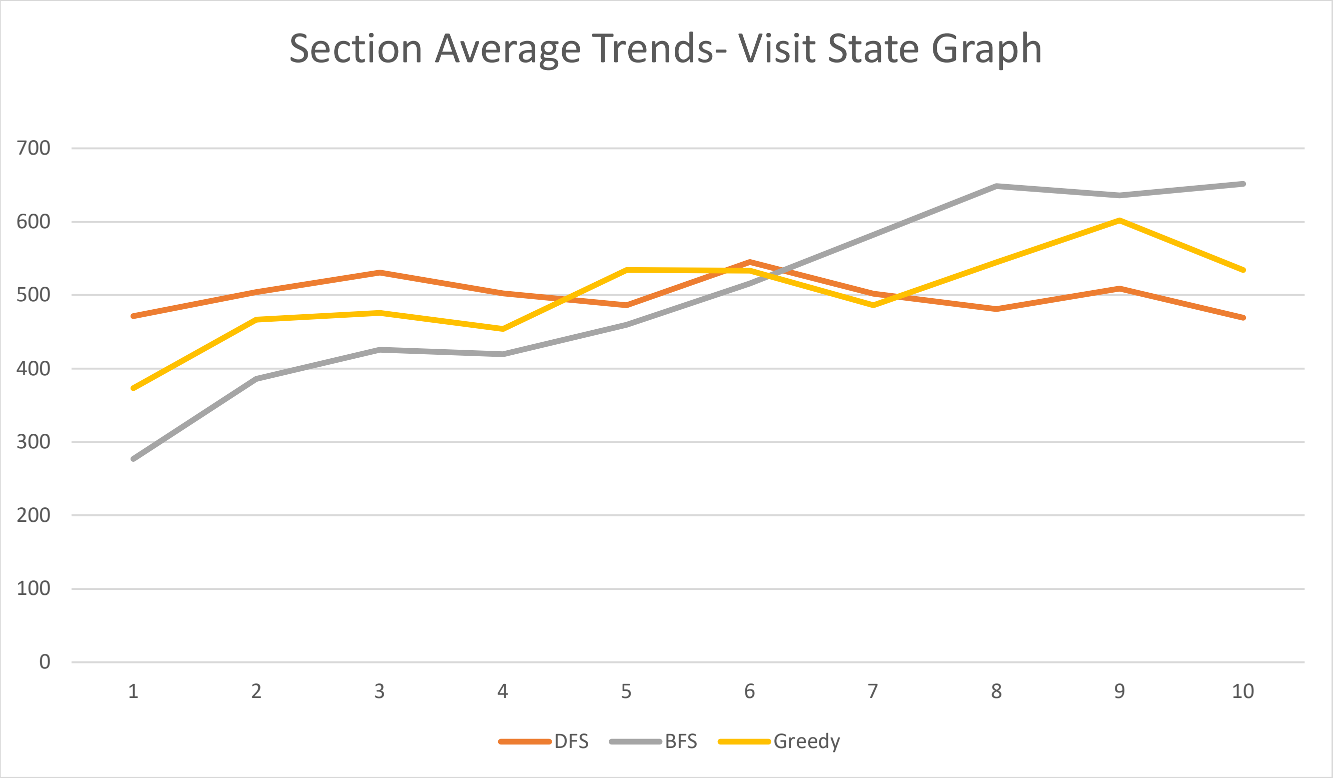
Average Depth Trends →



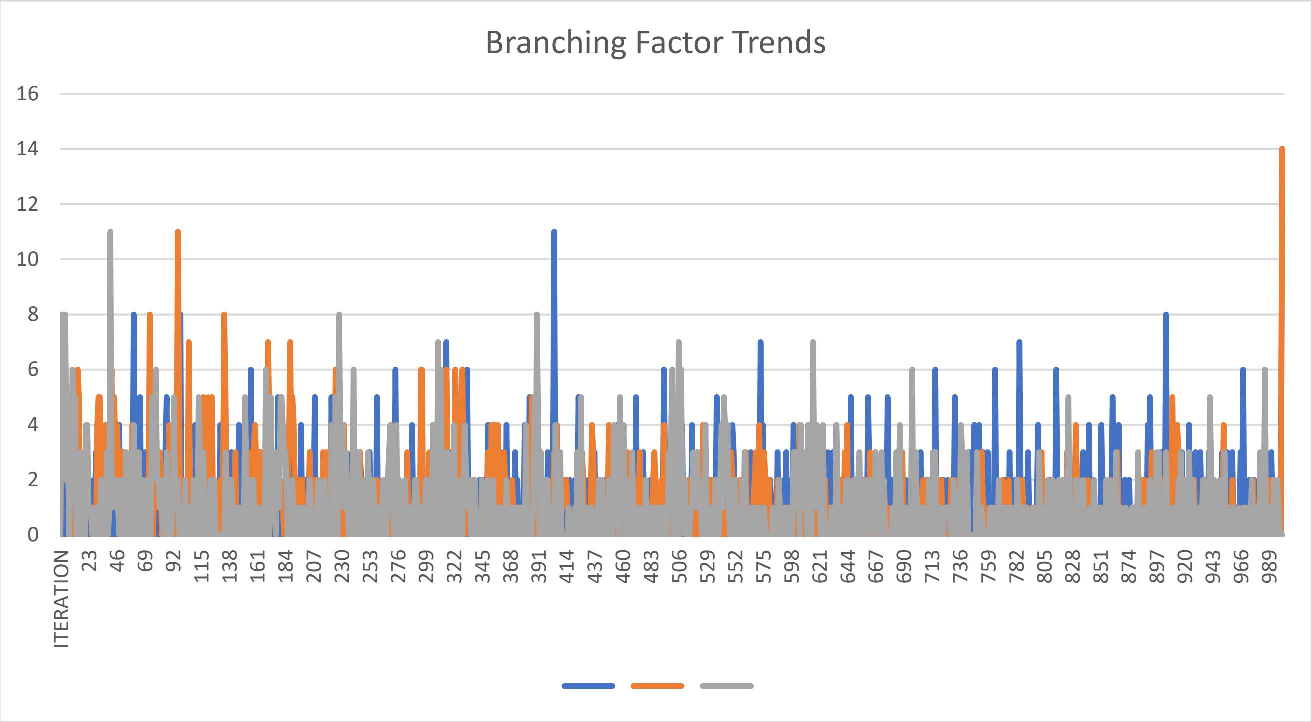


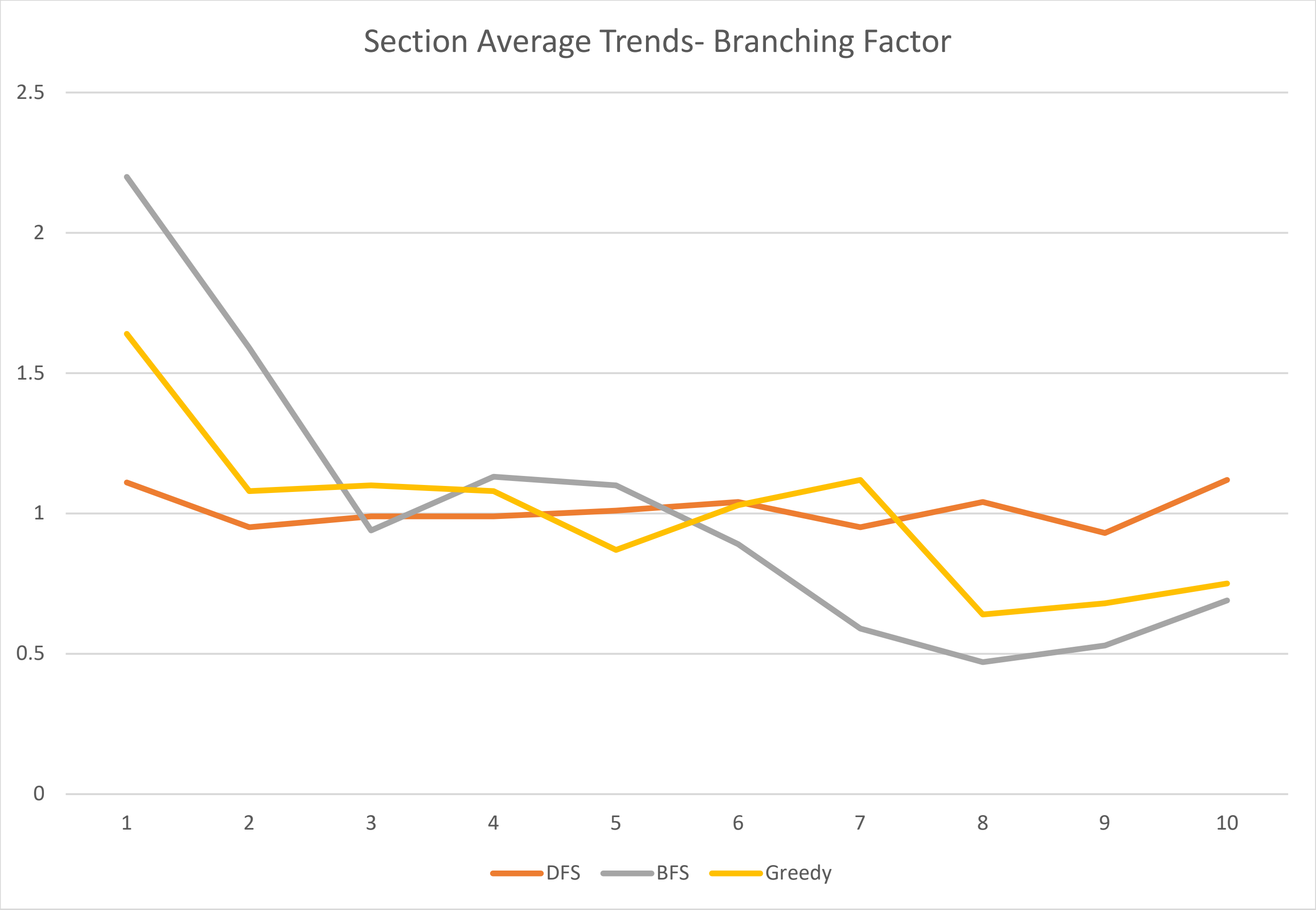
Visit State Graph →





Branching Factor Graph →


Branching Factor Trends
ITERATION


Section Average Trends- Branching Factor
DFS
BFS
Greedy