DFS vs BFS using Mathematical Computing

Kunal Suman[1]
and Lobsang Norbu[2]

[1] IIIT-D
[2] 2023295 kunal23295@iiitd.ac.in
[3] 2023303 lobsang23302@iiitd.ac.in

Abstract. This report analyzes DFS vs. BFS using mathematical computation and data visualization. We created 500,000 test cases using our graph creation algorithm, which uses different parameters to generate a graph. Our main finding, which we got, validates that factors like runtime and memory aren't directly proportional to the graphs but have multiple factors, which we have specified in the document. This clears many misconceptions regarding the nature and behavior of these algorithms.

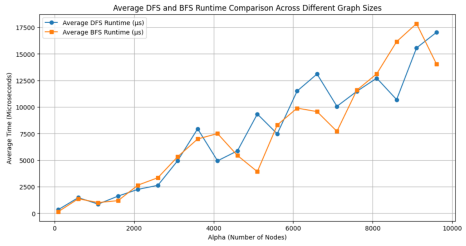
Keywords: Breadth-First Search (BFS) · Depth-First Search (DFS) · Erdős-Rényi (ER) · Barabási-Albert (BA) · Watts-Strogatz (WS) · Algorithm Performance · Random Graphs · Computational Complexity

1 Introduction

DFS and BFS are some of the most common and well-known traversal algorithms designed for unweighted graphs; our research focuses on exploring the relationship between graph size, density, run time, and total memory, mainly focusing on visualizing and studying data for directed as well as undirected graphs.

In our research, we figured out that the papers out there don't show their sampling data and how they created the test cases for their tests.[5].

There is also a conception that DFS performs better than BFS for more extensive graphs, which might not be accurate for every case; rather, it depends on multiple factors, including graph density, structure, and number of nodes[6]. Below is an example from our sample of our observation of undirected graphs, which contradicts the above statement.



2 DFS And BFS

2.1 DFS Overview

Dfs was initially studied by Charles Pierre Tremaux, a French mathematician, in 1900 to study mazes and find an optimal solution for solving these mazes[6].

Dfs searches deeper in the graph whenever possible, and it explores edges out of the most recently discovered vertex v that still has unexplored edges leaving it. Once all edges have been examined, the search backtracks will be used to explore edges leaving the vertex from which v was discovered.

```
DFS(G)
    for each vertex u ∈ G.V
        u.color = WHITE
        u.π =NIL
    time = 0
    for each vertex u ∈ G.V
        if u.color == WHITE
            DFS(G,u)

    DFS(G,u)
        time = time +1
        u.d = time
        u.color = GRAY
        for each vertex u ∈ G.Adj[u]
            if v.color == WHITE
                v.π = u
                DFS(G,u)
        time = time + 1
        u.f = time
        u.color = BLACK
```

Fig. 1. DFS Pseudo-Code

2.2 BFS Overview

Over the years, Bfs has gone through multiple iterations, creating an optimal graph traversal algorithm. It was initially invented in 1945 by Konrad Zuse as part of his Ph.D. thesis on Plankalkul[1]. It was reinvented and modified by Edward F. Moore in 1959, and C.Y Lee later developed BFS into a wire routing algorithm in 1961.

For a given graph with source vertex s, Bfs systematically explores the edges of G to discover every vertex that is reachable from s. It computes the distance from S to each reachable Vertex in G and creates a Bf_Tress with root s that contains all the reachable vertex[6].

```
BFS(G,s)
    for each vertex u ∈ G.V -{s}
        u.color = WHITE
        u.d = ∞
        u.π =NIL

    s.color = GRAY
    s.d = 0
    s.π = NIL
    s=Φ

    Enqueue(Q,s)
    While Q ≠ Φ
        u = Dequeue(Q)
        for each vertex v in G.Adj[u]
            if v.color == WHITE
                v.color = GRAY
                v.d = u.d + 1
                v.π = u
                Enqueue(Q,v)
        u.color = BLACK
```

Fig. 2. Enter Caption

3 Research Scope

BFS and DFS are some of the most well-known traversal algorithms used by various modern-day applications, from planning a city to designing a system for flight paths in aviation. One of the most common conceptions of these algorithms is that DFS always performs better than BFS for higher values of nodes, but this might not be true in every condition Iyanda, Joseph. (2023). Title: A Comparative Analysis of Breadth First Search (BFS) and Depth First Search (DFS) Algorithms[5]. It gives us a deep insight into these algorithms, but we figured out that there were no methods or test cases on which they concluded in the analysis part of their paper. We decided to validate their result by creating our test cases. Firstly, we have to get algorithms to create random graphs on which we will run our test cases. There are a lot of algorithms that exist and will work well, but we decided to make our own algorithm using probability and some mathematics

4 Existing Algorithms

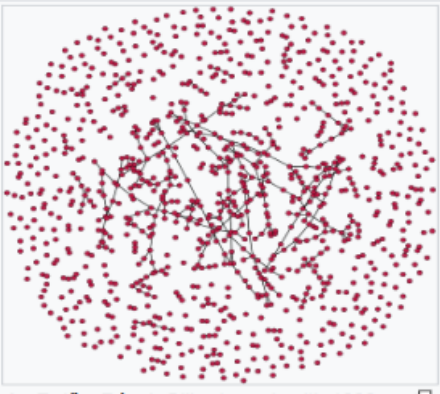
4.1 Erdős-Rényi (ER)

This model was created by Hungarian mathematicians Paul Erdos and Alfred Renyi in late 1950[2]. This model uses probability distribution to create random graphs, which are used to simulate real-world networks. There are two variants of the Erdős-Rényi model.

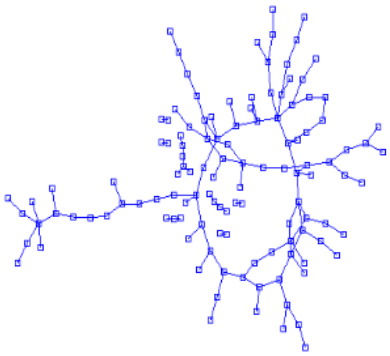
Model 1: We select a graph with n nodes and M edges from a pool of graphs with the same nodes and edges. It is denoted by G(n, M). In the G(3,2) Model,

there are three to two edges on three labeled vertices. In this, every graph has an equal chance of getting selected. So, it follows uniform distribution.

Model 2: In this model, a graph is generated by connecting label nodes. It is defined as G(n,p) with a probability of $P(G) = p^m(1p)^N m$. For a significant value of n, G(n,p) reaches binomial distribution, which is close to Poisson distribution $P(X = k) = ({}^k/k!)ek$, so the number of edges is not fixed and is calculated by probability.

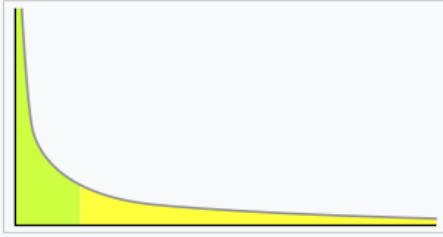


An Erdős–Rényi–Gilbert graph with 1000 vertices at the critical edge probability $p = 1/(n-1)$, showing a large component and many small ones

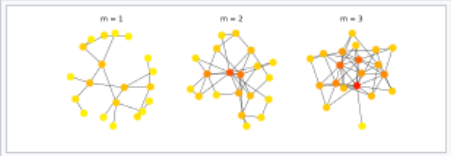


4.2 Barabási-Albert (BA)

This model was created by Albert-Laszlo Barabasi and Reka Albert in 1999[3]. It is used to generate random scale-free networks, and it follows a power law, which means the density of the graph is high even when the graph size is small. This method was created to overcome the limitations of Erdős-Rényi, which didn't use power law. Power law can be represented as $P(k)$ k-γ where k connection as γ is the parameter with values between $2 < \gamma < 3$. The probability that a new node is connected to another node is given by $pi = ki/ki$. This will add more connections, and this scaling effect is also known as the Matthew effect; this process runs till the size of N is reached. This model is used in a lot of biological experiments like metabolic pathways, drug interaction, protein synthesis, and molecular-level interactions. This model is used as the structure for modern applications.



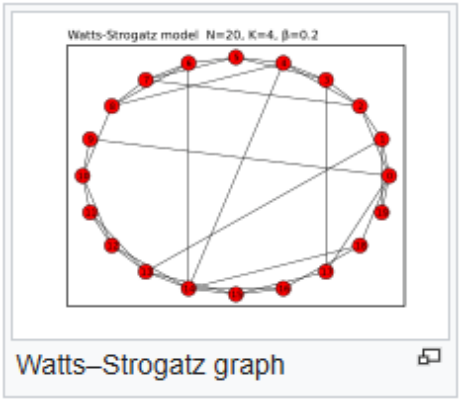
An example power-law graph that demonstrates ranking of popularity. To the right is the long tail, and to the left are the few that dominate (also known as the 80–20 rule).

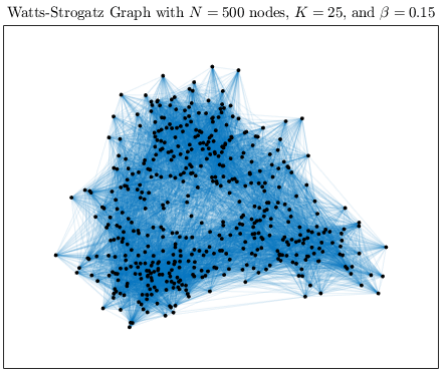


Display of three graphs generated with the Barabasi-Albert (BA) model. Each has 20 nodes and a parameter of attachment m as specified. The color of each node is dependent upon its degree (same scale for each graph).

4.3 Watts-Strogatz (WS)

This model was developed by Ducan J. Watts and Steven Strogatz in 1998[4]. This model generates random graphs with small-world properties (high density and low distances), usually used in social networking applications and real-time communication systems. This model generates undirected graphs with N nodes and NK/2 edges. A ring is created with N nodes and is connected to K/2 neighbors. Each edge and node in the graph has a probability β. 1st step produces a circular graph; when $\beta = 0$, no rewiring happens; if β =1, edges are rewired, and a new random graph is formed.





5 Methodology

Above are the few well-known graphing methods used all across the world, and most likely, the research papers on BFS vs. DFS published used one of these or

any other method, but we are unsure of their data collection method and test cases in which they have used to give such results.

Our primary focus is to validate the results given in the paper using mathematics and computing to prove the result for unweighted graphs.

A few keywords in our research method are:
α: Number of nodes in a graph
β: Number of nodes one node is connected to
γ: Inverse of the distance between 2 connected nodes
δ: Neighbours

In our example 1, we used the following graph generator, and the rest of the Bfs and Dfs code from above was followed.

```
for α in range ->10000,500
    β = 10
    G = Φ
    for i in range α
        G.add(i)
    for each α ∈ G.v
        connections = Φ
        while connection < β
            δ = rand()% α
            if δ ∉ connections
                add δ in connections
        for each δ in connections
            G.add(α,δ)
```

A sample of a few graphs which were generated using the above code is

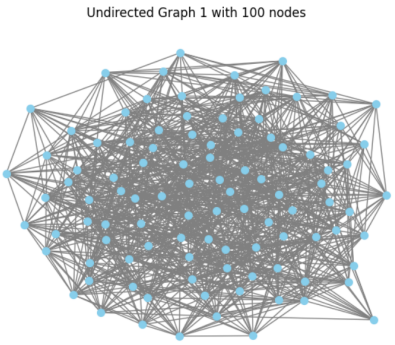


Fig. 3. Sample Graph 1

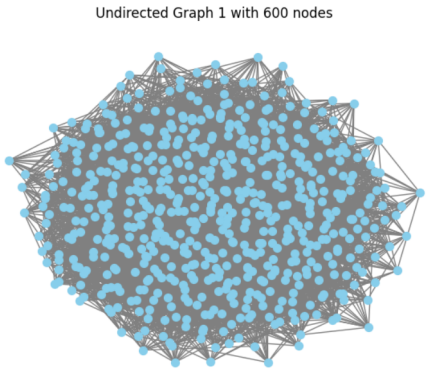


Fig. 4. Sample Graph 2

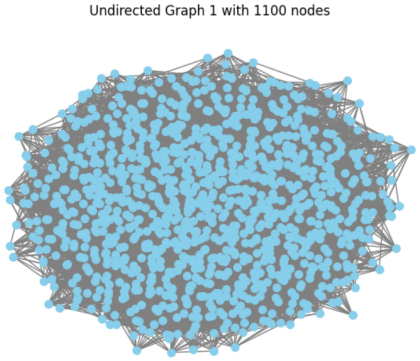


Fig. 5. Sample Graph 3

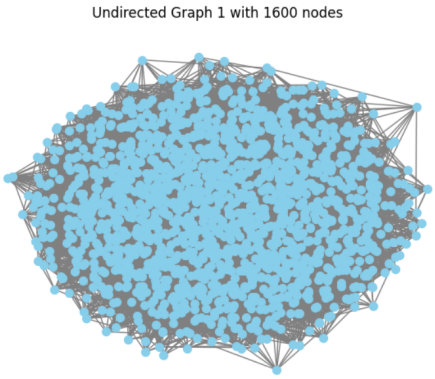


Fig. 6. Sample Graph 4

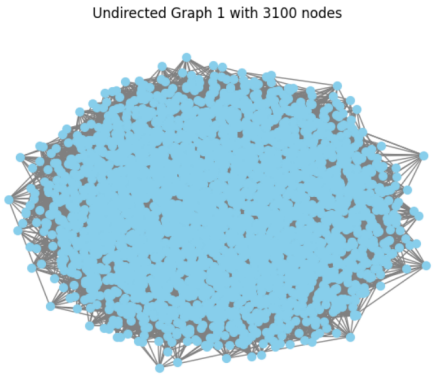


Fig. 7. Sample Graph 5

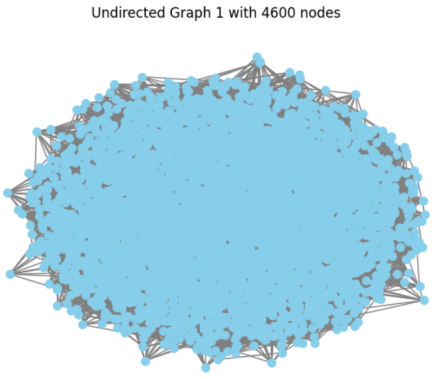


Fig. 8. Sample Graph 6

6 Our Results

The above cases are not enough to comment on the nature and run time as many parameters are fixed, so we will randomize the α, γ, and δ by using probability distributions to get more unbiased results and then figuring out and plotting our curve based on the output.

α is selected based on uniform probability distribution with a given fixed range, which we will change to simulate different graphs

β are the fixed values in a array [2,4,6,8,10]

γ is the inverse of the distance between the nodes

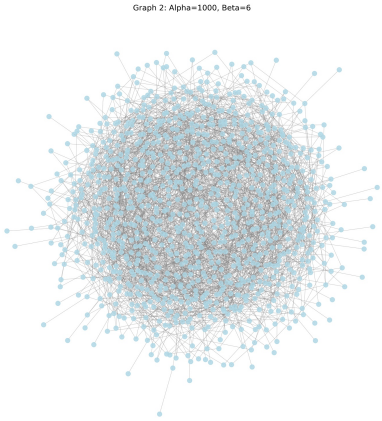
δ is the neighbors

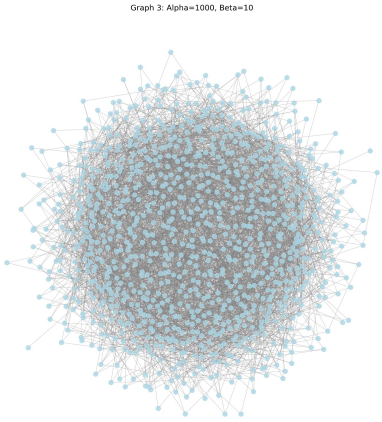
```
δ =[]
β =[2,4,6,8,10]
for i = 0 to 10000
    α = uniform(1000,20000)
    for j = 0 to 100
        for k = 1 to α
            for i = 1 to β
                γ = generate nodes using proximity
                δ.add(γ)
        file.out(α ,β ,δ)
```

Some of the sample graphs are

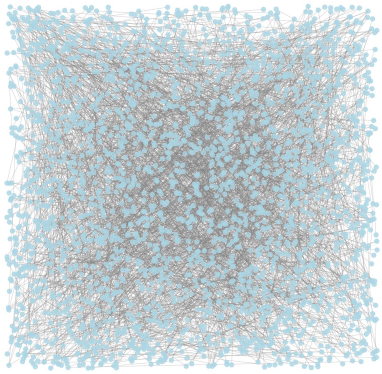
.


Graph 1: Alpha=1000, Beta=2

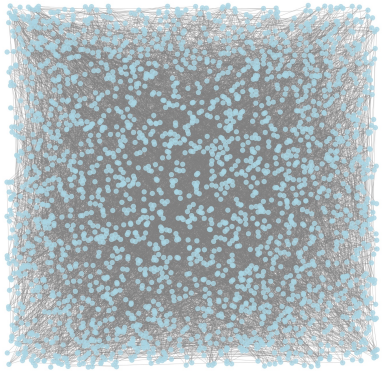

Graph 2: Alpha=1000, Beta=6


Graph 3: Alpha=1000, Beta=10

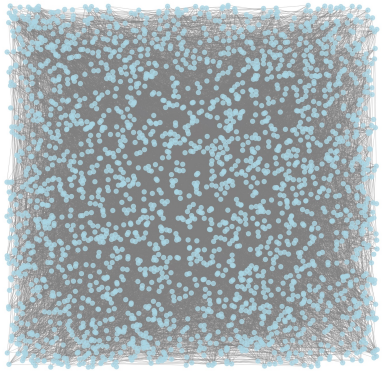
Graph 4: Alpha=2500, Beta=2



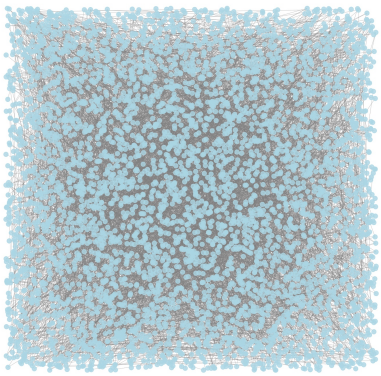
Graph 5: Alpha=2500, Beta=6



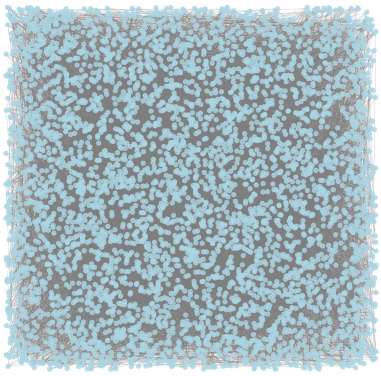
Graph 6: Alpha=2500, Beta=10



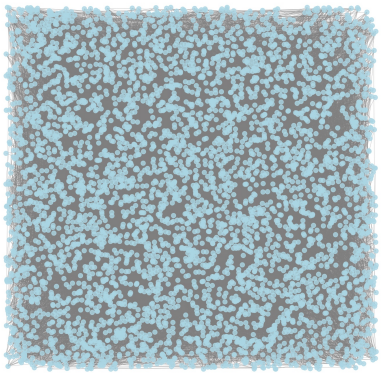
Graph 7: Alpha=5000, Beta=2



Graph 8: Alpha=5000, Beta=6

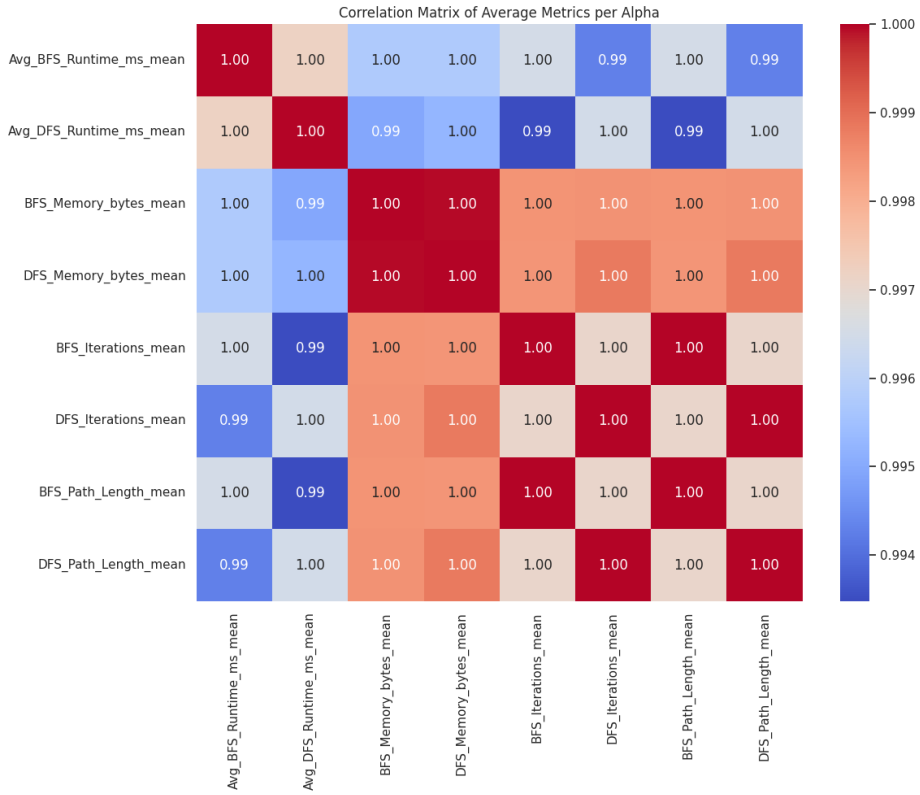


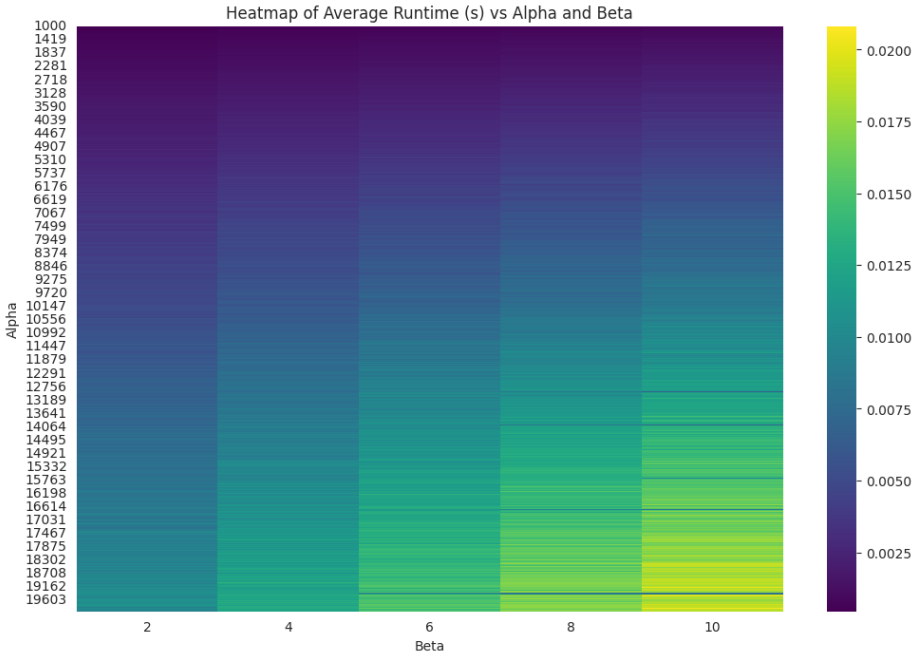
Graph 9: Alpha=5000, Beta=10

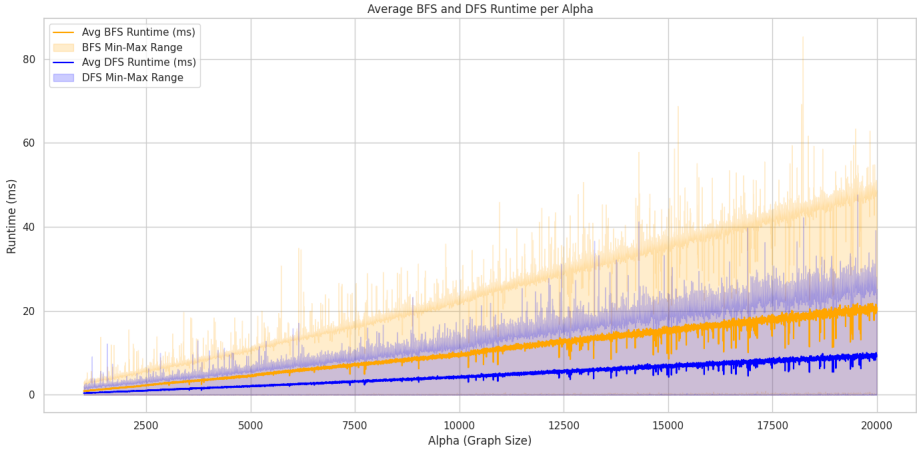


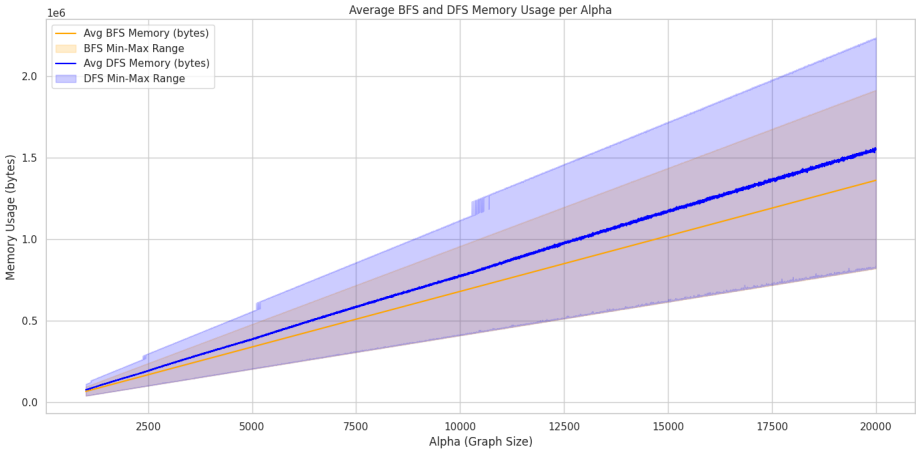
7 Output Data

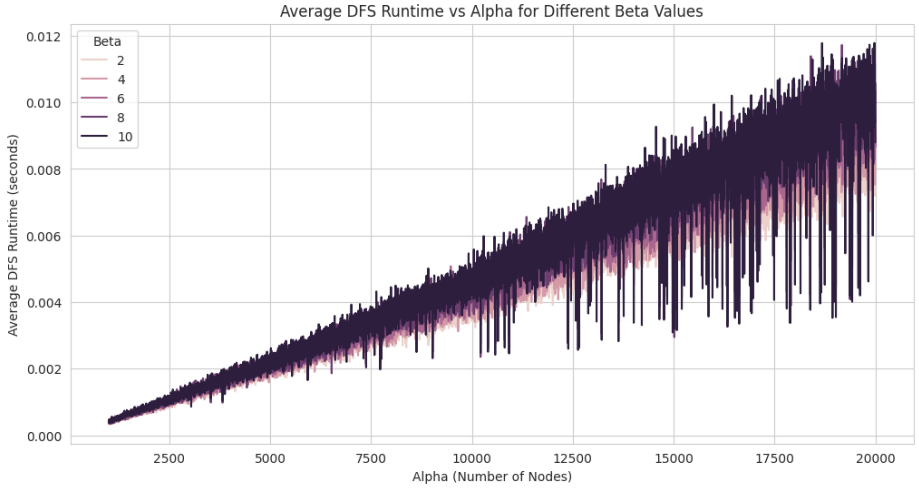
Below are some of the data in different forms of graphs that we got after compiling the code.Machine used is AMD Ryzen Threadreapper Pro 7995WX 96 Cores 192 Threads Some of them follow the trend, some don't, and others may seem like a fault value. when we look at the average values, there is an order, but if we go through it individually, there are a lot of variations that contradict the idea of dfs and bfs




Heatmap of Average Runtime (s) vs Alpha and Beta
Alpha
1000
1419
1837
2281
2718
3128
3590
4039
4467
4907
5310
5737
6176
6619
7067
7499
7949
8374
8846
9275
9720
10147
10556
10992
11447
11879
12291
12756
13189
13641
14064
14495
14921
15332
15763
16198
16614
17031
17467
17875
18302
18708
19162
19603
2
4
6
8
10
Beta
0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025


Average BFS and DFS Runtime per Alpha
Avg BFS Runtime (ms)
BFS Min-Max Range
Avg DFS Runtime (ms)
DFS Min-Max Range
Runtime (ms)
0
20
40
60
80
2500
5000
7500
10000
12500
15000
17500
20000
Alpha (Graph Size)


Average BFS and DFS Memory Usage per Alpha
1e6
Avg BFS Memory (bytes)
BFS Min-Max Range
Avg DFS Memory (bytes)
DFS Min-Max Range
Memory Usage (bytes)
0.0
0.5
1.0
1.5
2.0
2500
5000
7500
10000
12500
15000
17500
20000
Alpha (Graph Size)


Average DFS Runtime vs Alpha for Different Beta Values
Beta
2
4
6
8
10
Average DFS Runtime (seconds)
0.000
0.002
0.004
0.006
0.008
0.010
0.012
2500
5000
7500
10000
12500
15000
17500
20000
Alpha (Number of Nodes)

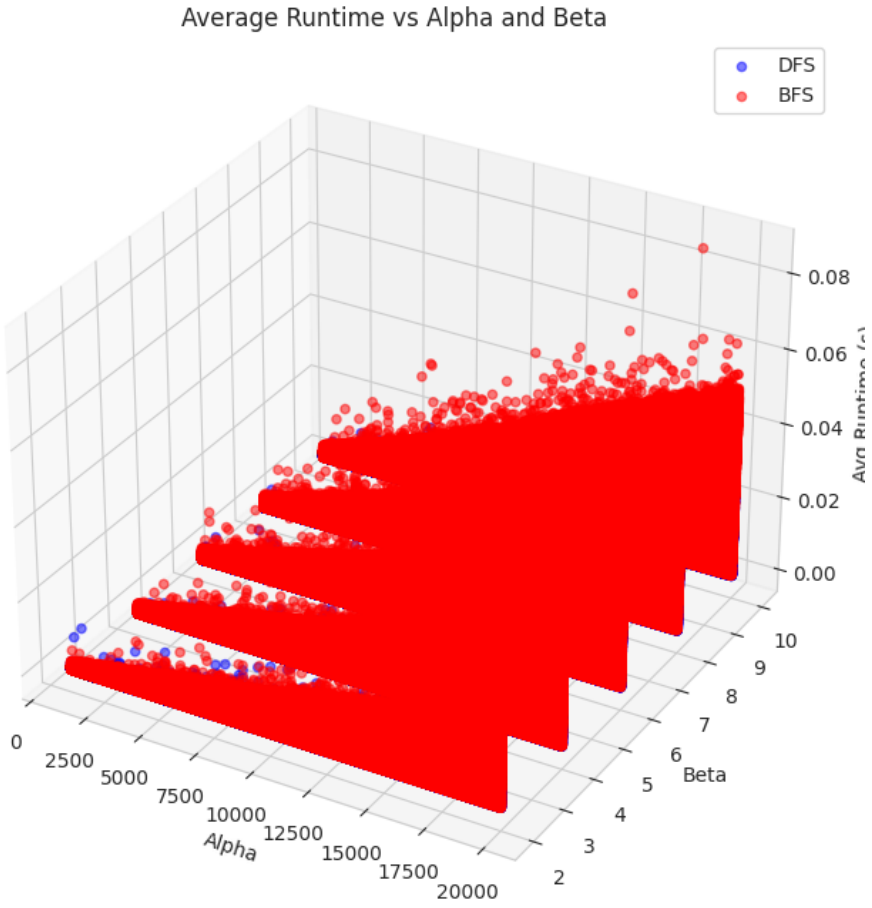
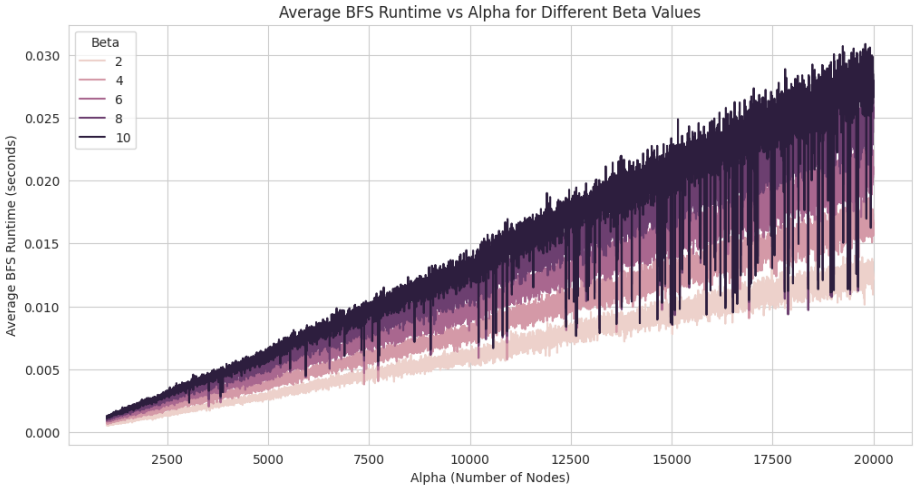
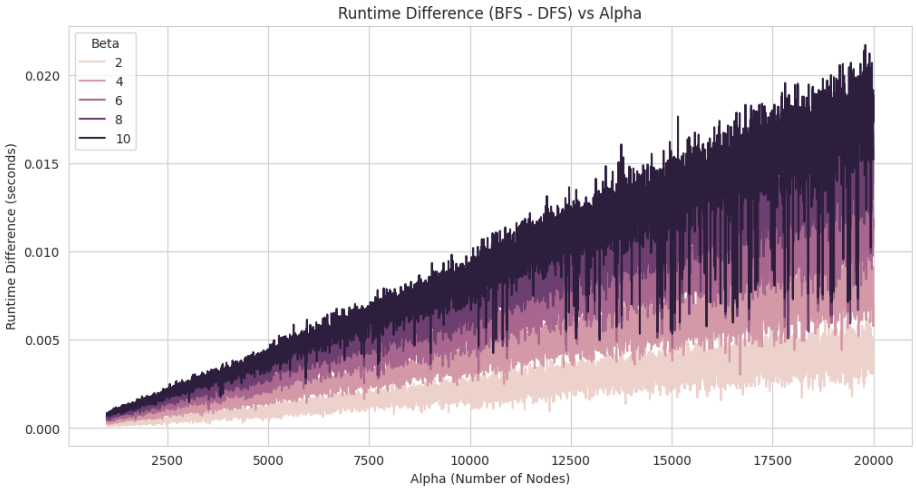
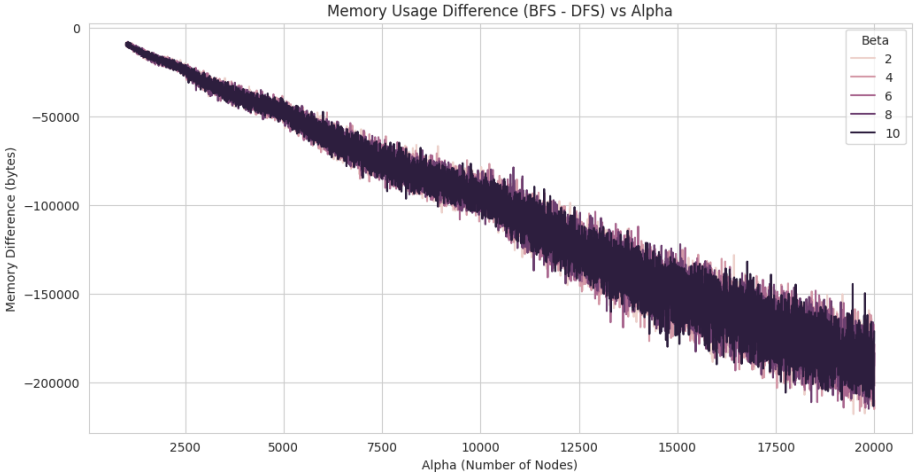
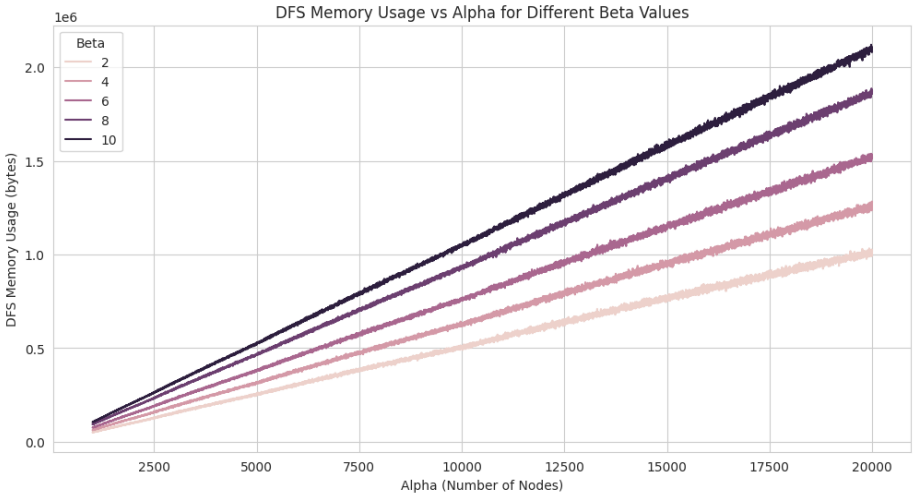


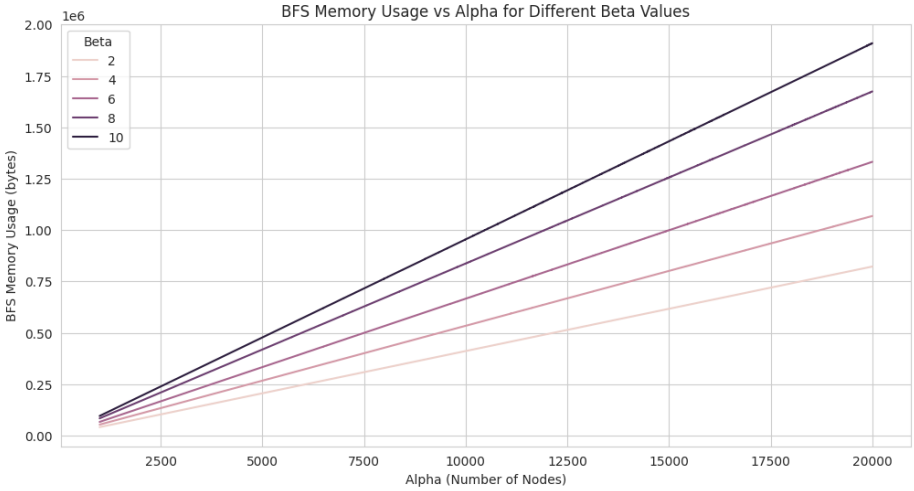
Fig. 9. Enter Caption

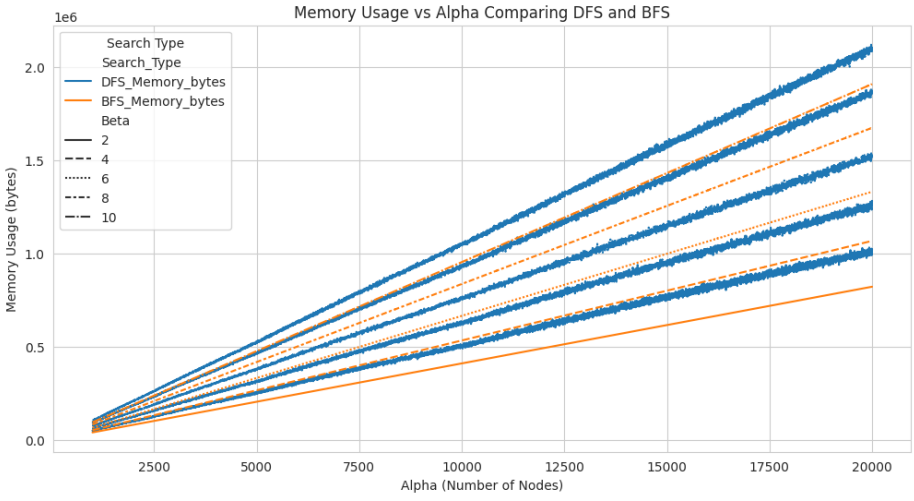

Average BFS Runtime vs Alpha for Different Beta Values
Beta
2
4
6
8
10
Average BFS Runtime (seconds)
0.030
0.025
0.020
0.015
0.010
0.005
0.000
2500
5000
7500
10000
12500
15000
17500
20000
Alpha (Number of Nodes)


Runtime Difference (BFS - DFS) vs Alpha
Beta
2
4
6
8
10
Runtime Difference (seconds)
0.020
0.015
0.010
0.005
0.000
2500
5000
7500
10000
12500
15000
17500
20000
Alpha (Number of Nodes)


Memory Usage Difference (BFS - DFS) vs Alpha
Memory Difference (bytes)
Alpha (Number of Nodes)
Beta
2
4
6
8
10
0
−50000
−100000
−150000
−200000
2500
5000
7500
10000
12500
15000
17500
20000


DFS Memory Usage vs Alpha for Different Beta Values
1e6
DFS Memory Usage (bytes)
Alpha (Number of Nodes)
Beta
2
4
6
8
10
2.0
1.5
1.0
0.5
0.0
2500
5000
7500
10000
12500
15000
17500
20000


BFS Memory Usage vs Alpha for Different Beta Values
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
BFS Memory Usage (bytes)
Beta
2
4
6
8
10
2500
5000
7500
10000
12500
15000
17500
20000
Alpha (Number of Nodes)


Memory Usage vs Alpha Comparing DFS and BFS
1e6
2.0
1.5
1.0
0.5
0.0
Memory Usage (bytes)
Search Type
Search_Type
DFS_Memory_bytes
BFS_Memory_bytes
Beta
2
4
6
8
10
2500
5000
7500
10000
12500
15000
17500
20000
Alpha (Number of Nodes)

8 Observations

8.1 Runtime and Memory

From the above graphs, a few relations can be figured out, like runtime increases with an increase in node and graph density, which is given by the heatmap, purple colour tells the time taken is less than bight yellow colour; some graphs might seem out of place. For example, our In our dfs vs. bfs run time analysis where bfs takes more time than dfs. That gap keeps on increasing with an increase in the number of α, and the same trends are observed in the average runtimes for each α; however, when we see another graph,i.e, between α and memory usage, we see a contradicting trend which tells us that memory usage of bfs is less than dfs. We know that at a larger scale, dfs becomes more memory efficient. What happened was that the graph wasn't really dense and more sparse, so this also validates why dfs was performing better than bfs in terms of runtime; that's why it took bfs longer time on average than does. In average DFS memory vs. alpha for Different beta values, we see the memory usage increases as α increases, which is a normal case, and we expect this behavior. However, in BFS Runtime vs Alpha for different Beta values, we have a contradictory graph as we expect BFS to perform better for higher values of α; this behavior could be due to variation in starting and ending nodes and node connection to other points, we can't say for sure what is the accurate reason behind this unless we see the undirected graph for ourselves.

8.2 Other Observations

With the help of the correlation Matrix, we got to know a few more trends, such as runtime is directly proportional to the number of iterations, which is to be expected, but there were cases when this rule didn't follow less number of iterations doesn't always mean less time. A few of the examples from our test cases are below. Our complete data is in

α	β	DFS Runtime	BFS Runtime	BFS Iterations	DFS Iterations
2232	2	0.386676	0.668497	263	534
8238	2	4.6535	5.47891	4721	6379
18489	10	16.6575	22.338	6581	14496
7654	6	6.63776	10.0958	5397	7464
14568	8	0.696677	1.2787	254	942

`https://drive.google.com/file/d/1RJ9fslXB54J9ZQNvfVVUT3bLZNPdJXjK/view`

9 Conclusion

Based on the above data we got, we can conclude that we can't really predict the runtime and memory usage of a graph using BFS or DFS. Multiple factors are

in play like number of nodes, connectivity of nodes, density, and the proximity of the start and end nodes. These factors affect the metrics: runtime, memory, and iterations of the two algorithms. Even though we see few observations that do support these type of trends-"DFS performs better at sparse graphs and BFS works better at dense graphs", it is not strictly followed always, and sometimes even the opposite may occur.

References

1. Zuse, K.: Plankalkül, Konrad-Zuse-Zentrum für Informationstechnik Berlin (1945)
2. Erdős, P., Rényi, A.: On Random Graphs I. Publications Mathematicae **6**, 290–297 (1959)
3. Barabási, A.L., Albert, R.: Emergence of Scaling in Random Networks. Science **286**(5439), 509–512 (1999).
4. Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world' networks. Nature **393**, 440–442 (1998).
5. Iyanda, J., et al.: A Comparative Analysis of Breadth-First Search (BFS) and Depth-First Search (DFS) Algorithms. International Journal of Computer Science and Network Security **23**(7), 55–60 (2023)
6. Meyer, U., Sanders, P.: Algorithms for Shortest Path Problems on Directed Graphs. Theory of Computing Systems **37**, 343–374 (2004).