STRUCTURE IN NUMBER THEORY: ARITHMETIC FUNCTION TREES

SAMVAR SHAH AND NISHKARSH SINGH

Abstract. The class of arithmetic functions (AFs) has been well studied for over a millennium, with pioneering contributions by mathematicians such as Erdos and Subbarao, who examined the iterates of these functions. In this paper, we introduce a novel structure—a generalized arithmetic tree (AFT)— which visualises these functions in the form of graphs. This representation makes it easier to study specific functions, such as Euler's Totient, the prime omega function, and the divisor function. We find an algorithm that computes the AFT in $\mathcal{O}(n \log n)$ time, a significant improvement over the naive $\mathcal{O}(n^2)$ method. We also prove a series of theorems on the structural and statistical properties of these trees, including results on layer averages, heights, and oscillatory behavior. Our work builds upon the classical studies of Erdős, Subbarao, and others, in iterative arithmetic functions

Mathematics is the queen of all sciences, and number theory is the queen of mathematics.

Carl Friedrich Gauss

Contents

1. Introduction

Definition 1.1 (Arithmetic Function)**.** In the context of our paper, a function $A : \mathbb{N} \to \mathbb{N}$ is arithmetic if and only if $A(n) < n$ for all $n > R$ for some $R \in \mathbb{N}$.

Our paper explores the transformation of arithmetic functions into graphical representations, which allow us to visualize and analyze properties.

Definition 1.2 (Arithmetic Function Tree)**.** Let A be an arithmetic function with fixed constant R. The arithmetic function tree (AFT) $G_A(N)$ (or simply G_A when N is implied) associated with A has R (typically 1) as its root, and each vertex V $(R < V \leq N)$ is connected to its parent $P(V) = A(V)$, forming an undirected edge.

This paper extends classical results on iterative arithmetic functions, particularly those studied by Erdős and M.V. Subbarao, who analyzed iterations of such functions.

The k-th iterate of A, denoted $A^k(n)$, is defined recursively as $A^k(n) = A(A^{k-1}(n))$ with $A^0(n) = n$. A naive construction of the AFT follows directly from iterating $A(n)$ for all $n \leq N$, establishing edges accordingly. However, this would take $\mathcal{O}(n^2)$ time.

Theorem 1.1. An AFT can be generated computationally from the arithmetic function A in $\mathcal{O}(n \log n)$ time.

Through the course of this paper, we find many and prove many theorems and lemmas regarding average degree of layers and heights of these sequences for Euler-Totient function, Prime Omega function and the $d(n)$ function. For instance we prove -

Theorem 1.2. If the *prime omega function* $\omega(n)$ is defined as the number of distinct prime divisors of n, $G_\omega(N)$ is defined as the graph we get from the AFT of the prime omega function, and $\deg_{G_\omega(N)}(A)$ is the degree of vertex A in that graph- then,
In $G_\omega(N)$,

$$\deg_{G_\omega(N)}(A) \sim \frac{N}{\log N} \frac{(\log \log N)^{A-1}}{(A-1)!}$$

for all $1 \leq A \leq N$.

Theorem 1.3. If γ_i be defined as the average of the degrees of all nodes in the i^{th} layer and $\{A_i\}$ is defined as the sequence where each A_i is the least number such that $H_\omega(A_i) = i$

$$\prod_{i=1}^{H(n)-1} \gamma_i = O\left(\frac{(N - A_{H_\omega(N)-1}) \log N}{N}\right)$$

Theorem 1.4. If the *Euler totient function* $\varphi(n)$ is defined as the count of integers less than or equal to n that are co-prime with n, then,
In $G_\varphi(N)$,

$$\log_2(V) + 1 \geq d(1, V) \geq \log_3\left(\frac{V}{2}\right) + 1$$

for all $1 \leq V \leq N$. where in any given tree, we define $d(U, V)$ as the distance between nodes U and V.

Another idea we dive into, is at what points the graphs of these arithmetic functions oscillate (i.e., change in their direction of propogation across layers) To anaylze this we created our very own Samkarsh Function

Definition 1.3.

$$\xi_{even}(x) = \sum_{p \leq x, p \equiv 1 (\text{mod } 4)} \ln(p)$$

$$\xi_{odd}(x) = \sum_{p \leq x, p \equiv 3 (\text{mod } 4)} \ln(p)$$

$$D(x) = \xi_{even}(x) - \xi_{odd}(x)$$

where $D(x)$ is the Samkarsh Function

which we analysed extensively using machine learning to find the exact points where this oscillating function changes in sign. The results of that and our computational evidence via the AFT have satisfactorily matched.

Since our paper does revolve on being able to produce AFT's for huge values of n a significant step is our novel algorithm, which reduces the time complexity of its generation to $\mathcal{O}(n \log n)$

(1) You can find the additional definition terms in Section 2.
(2) The Prime Omega deep dive is in Section 3.
(3) The Euler Totient deep dive is in Section 4.
(4) The Number of Divisors deep dive is in Section 5.
(5) For details about our algorithm, check out Section 6.
(6) Samkarsh is present in Section 7.

2. Additional Definitions

Definition 2.1 (Terms)**.** We define the following terms for our trees

(1) In any given tree, we define $d(U, V)$ as the distance between nodes U and V.
(2) We define the k^{th} layer of a tree as a collection of nodes V such that $d(R, V) = k$.
(3) We define $\nu_k(G)$ as the number of nodes in the k^{th} layer of the graph G.
(4) We define the height $H_A(N)$ of $G_A(N)$ as $\max_{i=1}^{N} d(R, i)$.
(5) For any node V in a graph G, $\deg_G(V)$ denotes the degree of V in G.

3. Prime Omega Function Tree

Definition 3.1. The *prime omega function* $\omega(n)$ is defined as the number of **distinct** prime divisors of n.

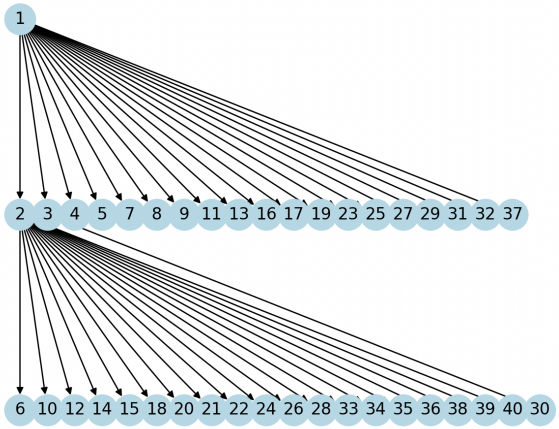


Figure 1. Example of a BFS Tree of a Prime Omega Function Tree for $n = 40$ nodes.

Lemma 3.1. In $G_\omega(N)$,

$$\deg_{G_\omega(N)}(A) \sim \frac{N}{\log N} \frac{(\log\log N)^{A-1}}{(A-1)!}$$

for all $1 \leq A \leq N$.

Proof. Note that this is equivalent to proving that,

$$\#\left\{\omega(x) = A;\ x \leq N\right\} \sim \frac{N}{\log N} \frac{(\log\log N)^{A-1}}{(A-1)!}$$

This is implied by proposition 8 of [4]. □

Now we will study the height of the tree, we can greedily observe that the height of the tree will only increase on primorial numbers, i.e. - $p_n\# = \prod_{i=1}^{n} p_i$ or the product of the first n primes.

Definition 3.2. $\{A_i\}$ is defined as the sequence where each A_i is the least number such that $H_\omega(A_i) = i$

Lemma 3.2. $A_n = \begin{cases} 2;\ n = 2 \\ p_{A_{n-1}}\#;\ n > 1 \end{cases}$

Proof. We can prove this via a greedy argument. We can observe that the smallest number in any layer would be a primorial, as any number smaller than a primorial will surely have less prime factors. Further, primorials will also be the generators of the smallest numbers of the next layer, as they will be the smallest number of prime factors a number would need to have to appear in the next layer. Combining the two observations, we can get the result. □

A_n grows extremely fast as it grows with the order of a tetration, by applying the prime number theorem, we can obtain that

Lemma 3.3.

$$A_n = \left({}^{n-1}\left(e^{(1+o(1))}\right)\right)^2$$

where ${}^b a$ denotes the b^{th} tetration of a.

Proof. We will first show that $\#p_n = e^{(1+o(1))n\log n}$

$$\begin{aligned}
\sum_{p\le x}(\log x - \log p) &= \sum_{p\le x}\int_p^x \frac{1}{t}\,dt = \int_1^x \pi(t)\frac{1}{t}\,dt, \\
\sum_{p\le x}(\log x - \log p) &= O\left(\int_1^x \frac{1}{\log t}\,dt\right) = o(x), \\
\sum_{p\le x}\log p &= \pi(x)\log x - \sum_{p\le x}(\log x - \log p) \\
\sum_{p\ge x} &= \log x\,(x\log x + o(x\log x)) - o(x) \\
&= x + o(x), \\
\sum_{k=1}^{n}\log p_k &= \sum_{p\le p_n}\log p = p_n + o(p_n),
\end{aligned}$$

We know by the PNT, that $p_n = n\log n + o(n\log n)$, so we have

$$\sum_{k=1}^{n}\log p_k = n\log n + o(n\log n)$$

Exponentiating both sides, we have $p_n = e^{(1+o(1))n \log n}$. As we know that $A_n = p_{A_{n-1}}$, applying this relation recursively, we get the result. □

n	A_n
1	2
2	6
3	30030
4	$> 10^{150000}$

TABLE 1. First 4 values of A_n

This is interesting because this shows that our tree's height grows extremely slowly, even though the density of $\lim_{x\to\infty} \frac{1}{x}\#\{n \leq x;\ \omega(n) = k\} = 0$ for all k, so our tree doesn't get 'bunched' asymptotically as $N \to \infty$.

Definition 3.3. A layer of a tree is a collection of nodes based on the distance of the nodes from the root. The i^{th} layer consists of all the nodes A with $d(1, A) = i$.

Definition 3.4. Let γ_i be defined as the average of the degrees of all nodes in the i^{th} layer.

Lemma 3.4.

$$\prod_{i=1}^{H(n)-1} \gamma_i \sim \frac{(N - A_{H_\omega(N)-1}) \log N}{N} = O(\log N)$$

Proof. Note that $\gamma_i = \frac{\sum_{d(1,V)=i} \deg_{G_\omega}(V)}{\nu_i(G_\omega)} = \frac{\nu_{i+1}(G_\omega)}{\nu_i(G_\omega)}$. Therefore,

$$\prod_{i=1}^{H(n)-1} \gamma_i = \prod_{i=1}^{H(n)-1} \frac{\nu_{i+1}(G_\omega)}{\nu_i(G_\omega)} = \frac{\nu_{H(n)-1}(G_\omega)}{\nu_1(G_\omega)}$$

We know that $\nu_1(G_\omega) \sim \frac{N}{\log N}$ by the prime number theorem. So we have,

$$\begin{aligned}\prod_{i=1}^{H(n)-1} \gamma_i &\sim \frac{(N - A_{H_\omega(N)-1}) \log N}{N} \\ &= O(\log N)\end{aligned}$$

□

Proof. Follows from the previous lemma. □

4. Euler Totient Function Tree

Definition 4.1. The *Euler totient function* $\varphi(n)$ is defined as the count of integers less than or equal to n that are co-prime with n.

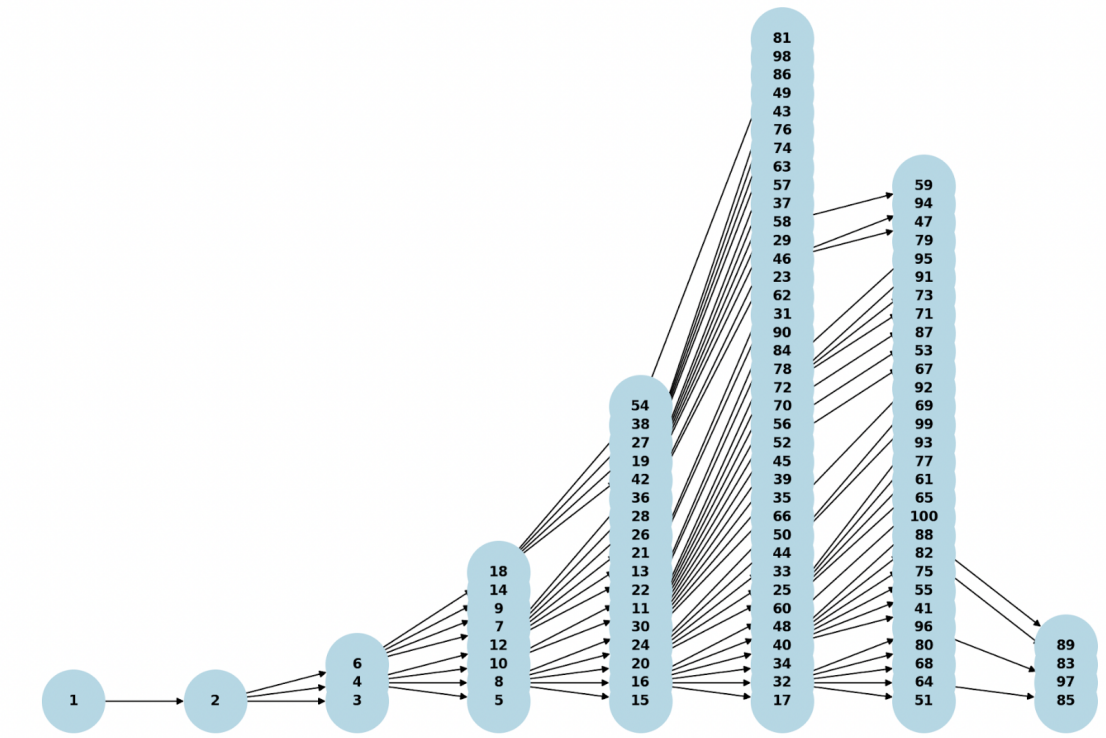


FIGURE 2. Example of a BFS Tree of an Euler Totient Graph for n = 100 nodes.

Lemma 4.1. In $G_\varphi(N)$,

$$\log_2(V) + 1 \geq d(1, V) \geq \log_3\left(\frac{V}{2}\right) + 1$$

for all $1 \leq V \leq N$.

Proof. We can observe that $\varphi^{d(1,v)}(v) = 1$. By Theorem 1 and 3 of [5], we get our result. □

Lemma 4.2. $d(1, V) = \log_2(V - 1) + 1$ iff V is a Fermat prime.

Further, we analyzed the frequencies of each value and after heavy computational evidence, we were able to pose the following conjecture

Conjecture 4.1. The frequency $f(k)$ of each totient value k in the Euler Totient Function Tree grows asymptotically as:

$$f(k) \sim \frac{N}{\zeta(2)} \cdot \frac{1}{k}$$

where $\zeta(2)$ is the Riemann zeta function at 2.

This implies that for large k, the number of nodes V with $\varphi(V) = k$ drops sharply, resulting in a logarithmic-like distribution where most totient values are relatively small.

Lemma 4.3. For each layer k, $\nu_k(G_\varphi(N))$ is constant for all $N \geq 2 \times 3^{k-1}$.

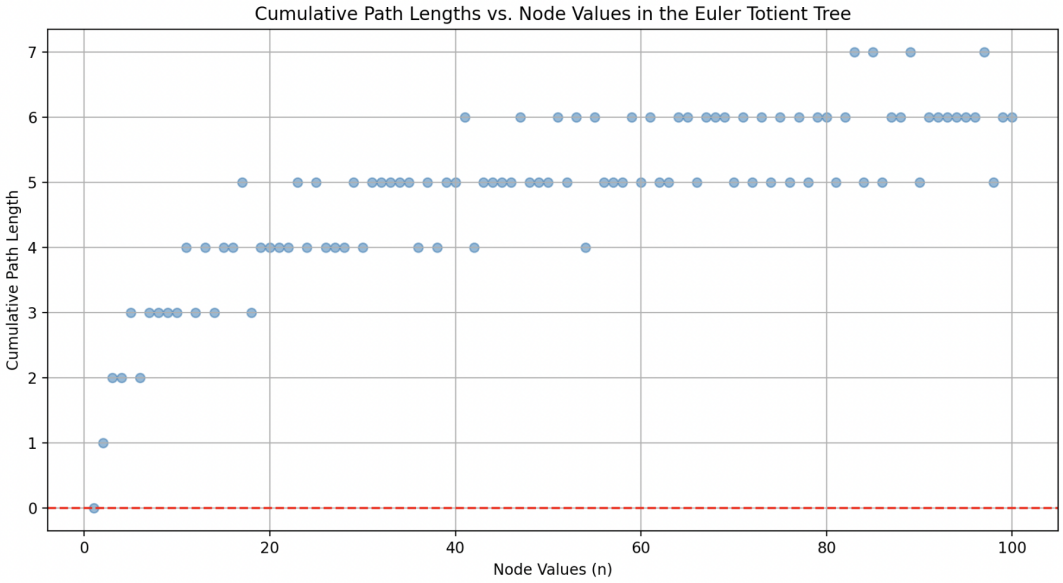


FIGURE 3. The relation between path lengths from source and node values (n_i)

5. Number of Divisors Function Graph

Definition 5.1. The $d(n)$ is defined as the number of integers $\leq n$ than which divide n.

Definition 5.2. We define the sequence $\{B_i\}$ such that B_i is the least number of nodes such that $H_d(B_i) = i$.

Definition 5.3. Define a function $\beta(n)$, let $n = q_1 q_2 q_3 \dots q_m$ be the prime factorization of n such that $q_1 \geq q_2 \geq \dots \geq q_m$. Let p_i denote the i^{th} prime, then $\beta(n) = p_1^{q_m - 1} p_2^{q_{m-1} - 1} \dots p_{m-1}^{q_1 - 1}$.

Conjecture 5.1. B_i behaves the following manner,

$$B_i = \begin{cases} 3; \ i = 1; \\ \beta(B_{i-1}); \ i > 1 \end{cases}$$

This conjecture has been tested till $B_i = 293318625600$ (or $i = 7$). This conjecture is also theoretically promising.

n	B_n
1	3
2	4
3	6
4	12
5	60
6	5040
7	293318625600

TABLE 2. First 7 values of b_n, computationally verified

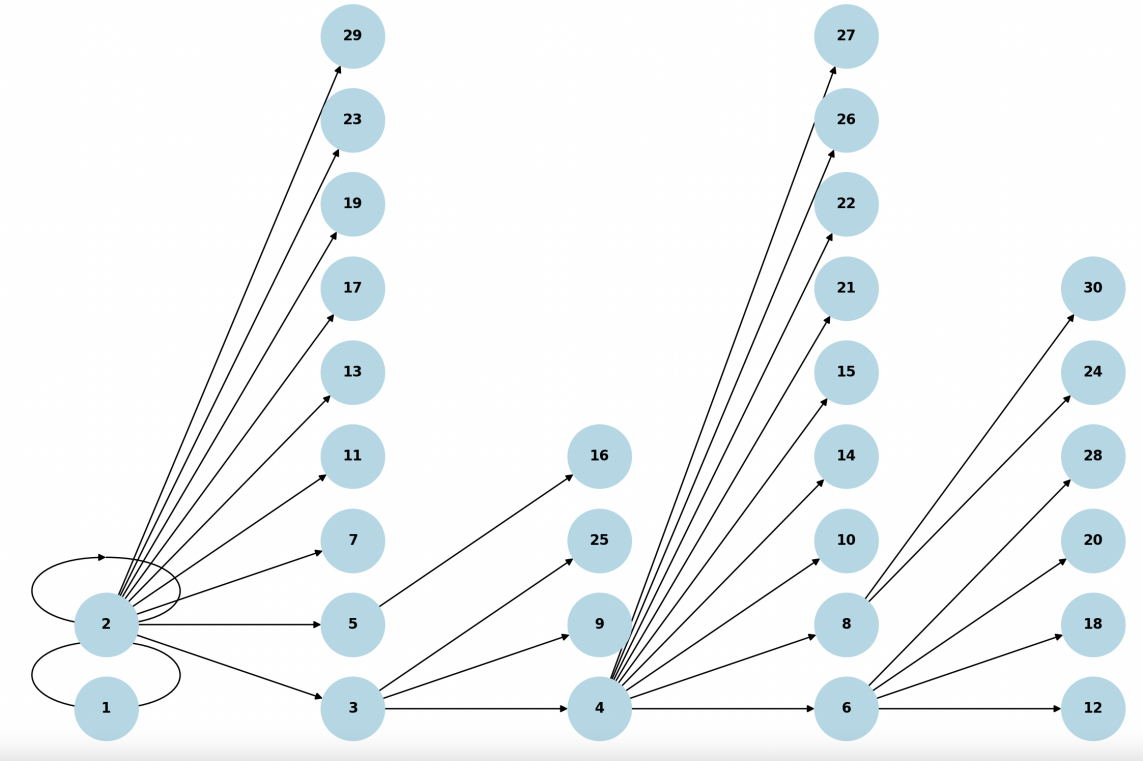


FIGURE 4. Example of a BFS Tree of the $d(n)$ graph for n = 30 nodes.

6. Coding the Graphs

6.1. **Algorithm.** The algorithm we use constructs an arithmetic function tree from an arithmetic function $A(n)$ in $\mathcal{O}(n \log n)$ time.

7. Coding the Graphs

7.1. **Algorithm.** The algorithm we use constructs an arithmetic function tree from an arithmetic function $A(n)$ in $\mathcal{O}(n \log n)$ time.

Algorithm 1 Constructing an Arithmetic Function Tree

Step 1: Compute Euler's Totient Function ($\phi(n)$)
Initialize an array ϕ where $\phi[i] = i$ for all i.
for $i = 2$ to n **do**
  if $\phi[i] = i$ **then** ▷ i is prime
    for $j = i$ to n step i **do**
      $\phi[j] = \phi[j] \times \frac{i-1}{i}$
    end for
  end if
end for
Step 2: Construct the Arithmetic Function Tree
Initialize an empty tree as an adjacency list (dictionary).
for $i = 1$ to n **do**
  Set $\phi(i)$ as the parent of i.
  Append i as a child of $\phi(i)$.
end for
Step 3: Perform BFS to Order the Tree
Initialize a queue with the root node (1).
while queue is not empty **do**
  Dequeue a node and process its sorted children.
  Assign levels based on BFS traversal.
end while
Step 4: Find Connected Components using DSU
Initialize Disjoint Set Union (DSU) with path compression.
for each (parent, child) pair in the tree **do**
  Union the parent and child.
end for
Find root representatives to determine connected components.
Step 5: Visualize the Tree
Construct a directed graph using NetworkX.
Assign positions based on BFS levels.
Draw the tree using Matplotlib.

7.2. **Proof of Correctness.** We aim to prove that three things being shown correct, proves the claim

- At every step, the invariant is maintained: if a vertex V is added to the tree, then it is connected with its parent $A(V)$. Since $A(V) < V$ for $V > R$, the parent property invariant is maintained.

- Because A is arithmetic, repeated application of A (i.e., following parent links) on any vertex V cannot return and lead to the formation of a cycle confirming that the graph remains a tree.
- The algorithm terminates after processing the entire Prüfer code and connecting the final two nodes. At this point, all vertices have been connected in a tree structure that satisfies the definition of an arithmetic function tree.

7.3. **Time Complexity Analysis.**

- Generating the Prüfer Code using a Fenwick Tree (BIT) takes $\mathcal{O}(n \log n)$ time. This is because each operation takes $\mathcal{O}(\log n)$ time, and we perform n operations, the total complexity is $\mathcal{O}(n \log n)$.
- A typical Sieve of Eratosthenes runs in $\mathcal{O}(n \log \log n)$, which is a lower order than $\mathcal{O}(n \log n)$.
- With path compression and union by rank, each DSU operation runs in nearly constant time, specifically $\mathcal{O}(\alpha(n))$ per operation, where $\alpha(n)$ is the inverse Ackermann function.
- Constructing the tree from the Prüfer Code using a Priority Queue would take $\mathcal{O}(n \log n)$ time due to the priority queue operations since each insertion and extraction from the priority queue takes $\mathcal{O}(\log n)$ time.

Since the dominant steps, Prufer code generation and tree construction, both take $\mathcal{O}(n \log n)$, and other steps are asymptotically smaller, the overall complexity of the algorithm is: $\mathcal{O}(n \log n)$

7.4. **Codes.** We provide the code used to buid the AFT for the euler totient function graph here

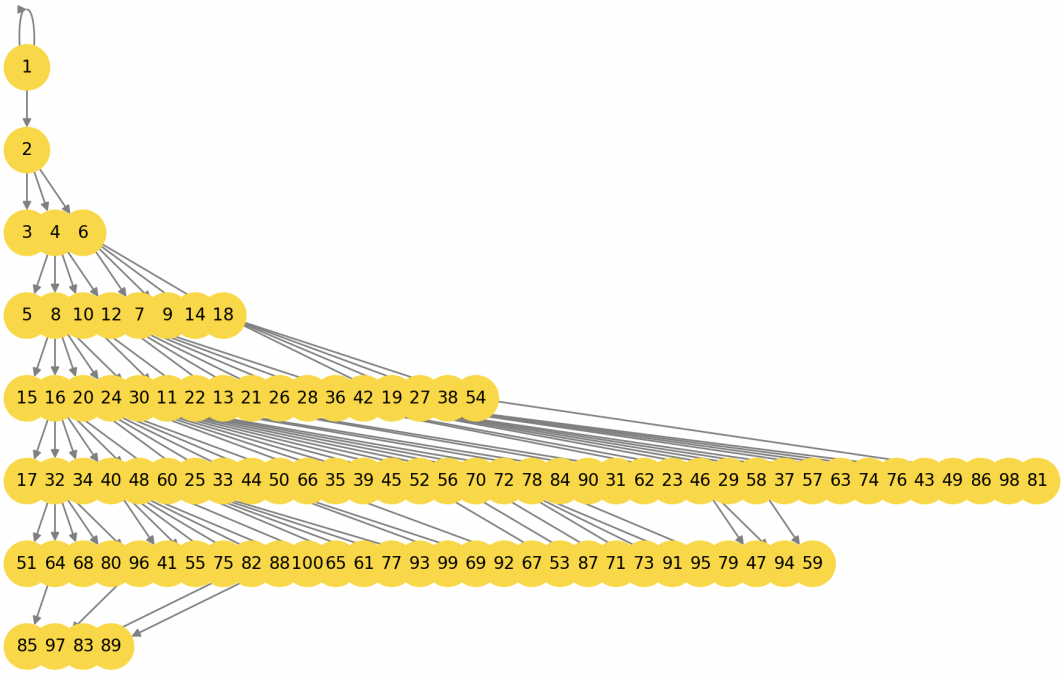


FIGURE 5. The optimum code's output graph.

The $\mathcal{O}(nlogn)$ takes 0.02 seconds to run while the $\mathcal{O}(n^2)$ takes 0.28 seconds a huge improvement (this is for a small value of n it gets much better for larger values of n.)

```
import matplotlib.pyplot as plt
import networkx as nx
from collections import deque


def sieve(n):
    phi = list(range(n + 1))
    for i in range(2, n + 1):
        if phi[i] == i:
            for j in range(i, n + 1, i):
                phi[j] *= (i - 1)
                phi[j] //= i
    return phi


def construct_arithmetic_function_tree(n, phi):
```

```
    tree = {}
    for i in range(1, n + 1):
        parent = phi[i]
        if parent not in tree:
            tree[parent] = []
        tree[parent].append(i)
    return tree


def bfs_tree_order(tree, root=1):
    queue = deque([(root, 0)])
    visited = set()
    ordered_tree = {}
    levels = {}

    while queue:
        node, level = queue.popleft()
        if node in visited:
            continue
        visited.add(node)
        levels[node] = level
        if node in tree:
            ordered_tree[node] = sorted(tree[node])
            for child in sorted(tree[node]):
                queue.append((child, level + 1))
    return ordered_tree, levels


def find_connected_components(tree, n):
    parent = list(range(n + 1))

    def find(x):
        if parent[x] != x:
            parent[x] = find(parent[x])
        return parent[x]

    def union(x, y):
        root_x, root_y = find(x), find(y)
        if root_x != root_y:
            parent[root_y] = root_x
```

```
    for node, children in tree.items():
        for child in children:
            union(node, child)

    components = {}
    for i in range(1, n + 1):
        root = find(i)
        if root not in components:
            components[root] = []
        components[root].append(i)

    return components


def visualize_tree(tree, levels, components):
    G = nx.DiGraph()
    for parent, children in tree.items():
        for child in children:
            G.add_edge(parent, child)

    plt.figure(figsize=(10, 6))
    pos = {}
    layer_widths = {}

    for node, level in levels.items():
        if level not in layer_widths:
            layer_widths[level] = 0
        pos[node] = (layer_widths[level], -level)
        layer_widths[level] += 1

    node_colors = ['gold' for _ in G.nodes()]

    nx.draw(G, pos, with_labels=True, node_color=node_colors, edge_color='gra
            arrows=True)
    plt.title("Arithmetic Function Tree with Connected Components")
    plt.show()


def main(n):
```

```
    phi_values = sieve(n)
    tree = construct_arithmetic_function_tree(n, phi_values)
    ordered_tree, levels = bfs_tree_order(tree)
    components = find_connected_components(tree, n)
    visualize_tree(ordered_tree, levels, components)
    return ordered_tree


n = 100
result_tree = main(n)
```

8. The Samkarsh Function

8.1. **Construction and Notes.**

Definition 8.1.

$$\xi_{even}(x) = \sum_{p \leq x, p \equiv 1 (\text{mod } 4)} \ln(p)$$

$$\xi_{odd}(x) = \sum_{p \leq x, p \equiv 3 (\text{mod } 4)} \ln(p)$$

$$D(x) = \xi_{even}(x) - \xi_{odd}(x)$$

Definition 8.2. In number theory, a quadratic residue modulo n is an integer that can be written as a^2(mod n) for some integer a

(1) Specifically, if we consider $n = 4$, then p is a quadratic residue modulo 4 iff $p \equiv 1$(mod n) and p is a non-quadratic residue modulo 4 if $p \equiv 3$(mod n)
(2) Chebyshev's bias refers to the observation that, in certain ranges of x primes of the form $4k+3$ may appear slightly more frequently than those of the form $4k+1$. This bias means that the sums $xi_{even}(x)$ and $xi_{odd}(x)$ may temporarily diverge due to the non-uniform density of primes in these residue classes for certain ranges.

8.2. **Results.**

- We can then differentiate $D(x)$ to study how quickly $\xi_{even}(x)$ and $\xi_{odd}(x)$ diverge or converge.
- The derivative $D'(x)$ represents the rate of change in the divergence between the sums of logarithms of primes in each residue class. If $D'(x) \geq 0$ then that means that $\xi_{even}(x)$ is growing faster than $\xi_{odd}(x)$. If negative the reverse is true.
- We find that the Integral of $D(x)$ from 2 to 1000: -16720.494916778112 which is negative. This means that
  (1) There are more contributions to the sum of logarithms from the primes congruent to 3 modulo 4 than from those congruent to 1 modulo 4.

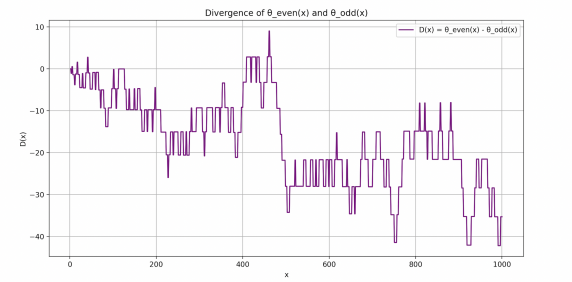


FIGURE 6. Divergence of $\xi_{even}(x)$ and $\xi_{odd}(x)$

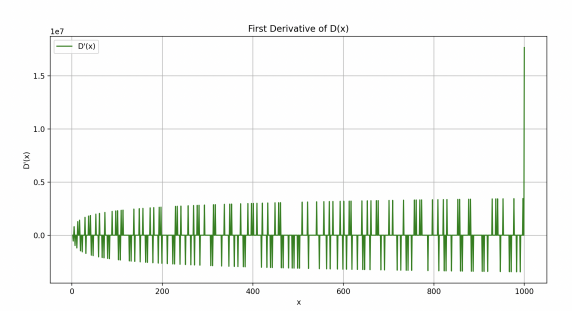


FIGURE 7. $D'(x)$

(2) This result is consistent with Chebyshev's bias with the negative integral suggesting that there is a significant bias towards primes of the odd class (3 modulo 4)

Definition 8.3. Let the cumulative sum $S(n)$ of $D(x)$ be defined as follows

$$S(n) = \sum_{x=2}^{n} D(x)$$

Definition 8.4. Let the k^{th} moment of $S(n)$ be defined as follows

$$M_k(n) = \frac{1}{n} \sum_{m=1}^{k} S(m)^k$$

Lemma 8.1. Moments of $S(n)$

(1) First moment (mean)

$$M_1(n) \to E[S(n)] \approx -C$$

for some constant $C > 0$

(2) Second moment (variance)

$$Var(S(n)) = M_2(n) - (M_1(n))^2 \implies M_2(n) \gg (M_1(n))^2$$

(3) For $k >= 3$

$$Skewness = \frac{M_3(n)}{(M_2(n))^{\frac{3}{2}}}$$

$$Kurtois = \frac{M_4(n)}{(M_2(n))^2} - 3$$

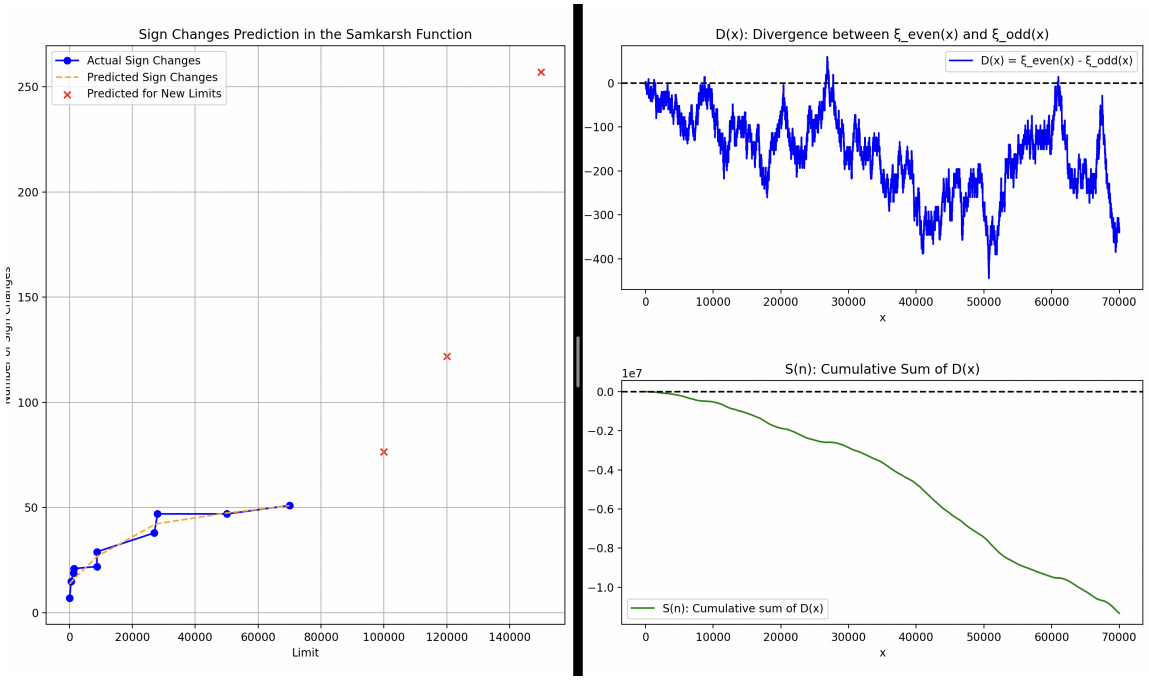
By computation we get that

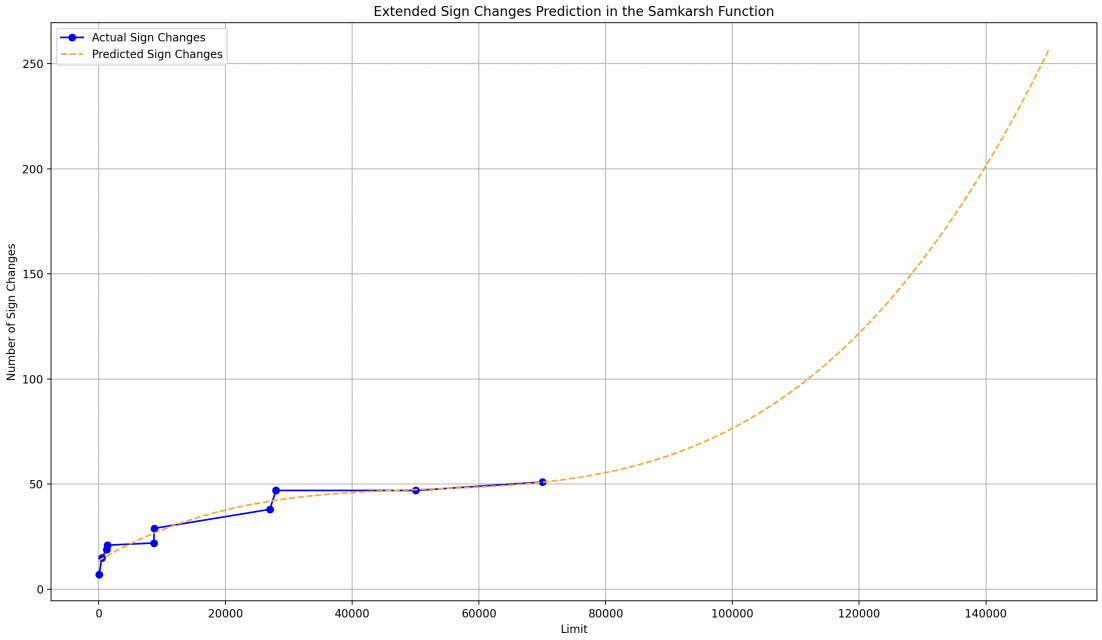
- Mean of S(n): -6280.921965563416
- Variance of S(n): 24904329.294835463

8.3. **Graphical Relation of Samkarsh Function.**

Definition 8.5. Let $D*_i$ be the number of times the sign of the $D(x)$ function changes uptil the limit value i.

We build a predictive model to analyze the exact point at which the oscillating function $D(x)$ changes its sign. We find that


Sign Changes Prediction in the Samkarsh Function
Actual Sign Changes
Predicted Sign Changes
Predicted for New Limits
Number of Sign Changes
Limit
D(x): Divergence between ξ_even(x) and ξ_odd(x)
D(x) = ξ_even(x) - ξ_odd(x)
x
S(n): Cumulative Sum of D(x)
1e7
S(n): Cumulative sum of D(x)
x


Extended Sign Changes Prediction in the Samkarsh Function
Actual Sign Changes
Predicted Sign Changes
Number of Sign Changes
Limit

Theorem: Let $P(n)$ be the set of all primes $p \leq n$, and let $R_1(n)$ be the set of primes congruent to 1 mod 4 and $R_3(n)$ be the set of primes congruent to 3 mod 4. Construct a bipartite graph $B(n)$ such that:

- **Vertices**: The vertices of the graph $B(n)$ are the sets $R_1(n)$ and $R_3(n)$.
- **Edges**: An edge is formed between a vertex $r_1 \in R_1(n)$ and a vertex $r_3 \in R_3(n)$ if and only if the quadratic residue condition is satisfied, meaning there exists an integer k such that $r_3 \equiv k^2 \mod r_1$.

Result: The number of edges $E(B(n))$ in the bipartite graph $B(n)$ is proportional to the sign changes in the Samkarsh function $D(n) = \xi_{\text{even}}(n) - \xi_{\text{odd}}(n)$ as $n \to \infty$.

Conjecture 8.1. : The relationship can be approximated as:

$$D*_n \sim C \cdot |E(B(n))|$$

for some constant $C > 0$.

Acknowledgements

We would like to thank Prof. Eric Riedl and Prof. Amanda Serenevy for their mentorship throughout this research. Their guidance and meticulous review of our mathematical work were instrumental in shaping this paper.

References

[1] Régis de La Bretèche and Gérald Tenenbaum. Remarks on the selberg–delange method. *arXiv preprint arXiv:2010.12929*, 2020.

[2] P Erdös and MV Subbarao. On the iterates of some arithmetic functions. In *The Theory of Arithmetic Functions: Proceedings of the Conference at Western Michigan University, April 29–May 1, 1971*, pages 119–125. Springer, 2006.

[3] Adolf Hildebrand and Gérald Tenenbaum. On the number of prime factors of an integer. 1988.

[4] Edmund Landau. Sur quelques problèmes relatifs à la distribution des nombres premiers. *Bulletin de la Société Mathématique de France*, 28:25–38, 1900.

[5] S. S. Pillai. On a function connected with $\$\phi\,(n)\,\$.Bulletinof theAmericanMathematicalSociety, 35 : 837 - -841, 1929.$