The Sum and Product Game: Longest Chains and Other Counts

Mariam Abu-Adas

December 2022

Abstract

In this project we work with the graph of the sum and product game and aim to find countings of various elements of the graph. We count the number of sum nodes and the distribution of their degrees. We also estimate the number of product nodes and use a Markov chain to estimates the distribution of their degrees. We use these countings and estimates to develop a probability proof on the parity of chain lengths.

1 Introduction

The Sum and Product Puzzle was originally published in 1969 by Hans Freudenthal and was popularized in 1979 by Martin Gardner who coined the name "The Impossible Puzzle". Many variations of the game then sprung up in the following years by various mathematicians. Our summer research focused on understanding the game from a graphical perspective which allowed us to address any one of the versions of the game, both those that can and can't be solved. In this paper, we work with the graph closely, understanding the correlation between the graph features and the solvability of the puzzle. Most of the paper aims to develop estimate countings of nodes and degrees of nodes in order to gather the necessary information needed for a probability proof. The proof is one that aims to justify the patterns in the parity of the game's length.

2 The Sum and Product Game

Two numbers k and l are selected privately such that $2 \leq l \leq k \leq 10$. The sum of the two numbers $k + l$ is given to the individual Sum, and the product kl is given to the individual Product. The Observer is a third individual with no knowledge of the numbers, their sum or their product. It is easy to see that given specific k and l values, Sum or Product may already know the value of the two numbers. For example, if $k = 5$ and $l = 2$, then while Sum would not immediately know what l and k are, Product would know the values immediately, for their is only one factorization of the number 10 (recall that k, l cannot equal 1). So a conversation between the two may go as follows:

Sum: I do not know your product
Product: I do know your sum
Sum: Now I know your product

In knowing that Product was able to discern the values of k and l, Sum was then able to figure out their values as well. However, knowledge that the other does *not* know the values of k and l is still knowledge that can assist either Sum or Product in figuring out the numbers. So the conversation can also go as follows:

Sum: I do not know your product
Product: I do not know your sum
Sum: I still do not know your product
Product: I still do now know your sum
Sum: Now I know the product
Product: Now I know your sum

In some cases, the Observer may also discern the values of l and k simply in hearing the exchange between Sum and Product, however this is not always the case.

Given specific l and k values, it is also possible for both Sum and Product to never discover the value of l and k. Their conversation would go on infinitely where both would say they still don't know the values of l and k.

The game may also be played with larger values of l and k. We denote n as the largest possible value, so in the above example, we let $n = 10$, however in general, we can define any n value and we let $2 \leq l \leq k \leq n$.

3 The Graph

The sum and product game can be depicted using a graph as shown below. For every pair l and k, we define an edge connected to two nodes: a pink node with the sum $l + k$ and a blue node with the product $l * k$ (in other graphs we may depict the product node by a square, and the sum node with a circle). The graph does not include the values of l and k for readability.

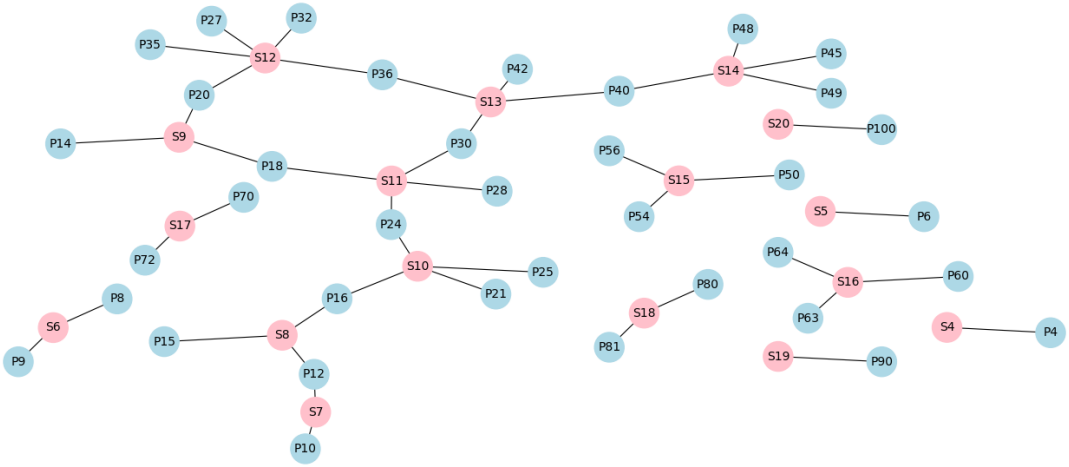


Figure 1: Graph of sum and product game for $n = 10$

4 Notation

Some notation for future use:

$$
\begin{aligned}
G(n): &\ \text{Graph with max value } n \quad (2 \leq l \leq k \leq n) \\
V_p(n): &\ \text{Product nodes in } G(n) \\
V_s(n): &\ \text{Sum nodes in } G(n) \\
d(V(n)): &\ \text{Degree of node in } G(n) \\
V_{p,d=3}(n): &\ \text{Condition on node (product node with degree equal to 3).}
\end{aligned}
$$

5 Counts

We begin by developing counts for various elements of the graph to use in upcoming proofs.

5.1 Number of Edges

An edge depicts a pair l and k whose values are between 2 and n. The number of edges is then equal to the number of ways we can select two numbers between 2 and n allowing for repetition (as l and k can equal each other). Thus the number of edges is equal to:

$$\binom{(n-1)+2-1}{2} = \frac{n(n-1)}{2}$$

5.2 Degree of Given Sum Node

To count the degree of a sum node, we partition the sum as evenly as possible, and find the difference between the partition and the lower and upper bounds of 2 and n. So for example, for the sum node 15, we first consider the most even partition, $l = 7, k = 8$. We count the

number of times we can decrease l and increase k until one or the other hits the bounds. That is, we calculate $7-2$ and $n-8$, and the degree of the sum node will be one more than the minimum of these two values. Therefore the degree of a sum node can be given by the following minimum function:

$$\min(n - \left\lceil \frac{V_s}{2} \right\rceil + 1, \left\lfloor \frac{V_s}{2} \right\rfloor - 1)$$

5.3 Degree of Given Product Node

Product nodes are more difficult to work with as we are dealing with the number of ways one can factor a number (not including the trivial factorization). As of now, we have a count for the final degree of a product node, as in, what the degree of a product node is given that n is big enough to encompass all possible factorization. To do this we must use the prime factorization of a number. So we write a product node V_p as the product of primes $p_1^{a_1} p_2^{a_2} \dots p_m^{a_m}$. Then the number of ways you can partition these primes into two non-empty sets is defined by

$$\left\lceil \frac{\prod_{i=1}^{m}(a_i+1)}{2} \right\rceil$$

5.4 Number of Sum Nodes

The number of sum nodes is an easy count. The smallest sum node possible is $2+2=4$ and the largest sum node is $n+n=2n$. It is clear to see that every value between 4 and $2n$ is achievable as a sum node, thus the total number of sum nodes is

$$2n-3$$

6 Estimates

There are some properties of the graph that we are unable to count with accuracy, but we can develop estimates for.

6.1 Estimate for Number of Product Nodes

To estimate the number of product nodes in our graph we consider the range of product nodes possible, then subtract the sets of products that are not achievable.
The smallest product node possible is $2*2=4$. The largest product node is $n*n=n^2$. We define 3 cases of values that are not included in our product sets.

1. Primes between 2 and n^2
   (a) Case 1: Primes less than and equal to n
   (b) Case 2: Primes between n and n^2
2. Multiples of primes that fall under case (b) above

3. Product of three primes such that every partition of the primes contains a value larger than n

We wish to count the number of values that fall under each of the above cases. We group case 1b with case 2 for convenience.

1. Number of primes less than or equal to n:

Using the prime number theorem, we know that the density of primes less than n is $\frac{1}{\ln(n)}$, thus the number of primes less than or equal to n can be approximated by the function

$$\pi(n) := \frac{n}{\ln(n)}$$

2. Number of primes and their multiples between n and n^2:

We use the prime density function we mentioned above. We integrate between n and n^2 over primes p with the function $\frac{1}{\ln(p)} * \frac{n^2}{p}$, which is the density of primes times the number of multiples of the primes between n and n^2.

$$\int_n^{n^2} \frac{1}{\ln p} * \frac{n^2}{p} dp \;=\; n^2(\ln 2)$$

2. Number of triples such that each partitioning contains a value larger than n

An example of this for $n = 10$ is the value 75, which has prime factorization $3 * 5 * 5$. The only possible partitions are $15 * 5$ or $3 * 25$, an in both cases one number is larger than 10, and hence 75 cannot exist as a product node in the graph of $G(10)$.

Let x, y, z be three primes between 2 and n. To count the number of triples, we need to count the number of values that satisfy the following conditions:

$$\begin{aligned} xyz &\leq n^2 \\ xy &> n \\ xz &> n \\ yz &> n \end{aligned}$$

In order to find the solution space to the above system of equations, we work in the log-log-log space to make our equations linear. We define the following variable changes:

$$\begin{aligned} \tilde{x} &= \ln(x) \\ \tilde{y} &= \ln(y) \\ \tilde{z} &= \ln(z) \\ \tilde{n} &= \ln(n) \end{aligned}$$

Using these new variables we have the following system of equations:

$$\begin{aligned} \tilde{x} + \tilde{y} + \tilde{z} &\leq 2\tilde{n} \\ \tilde{x} + \tilde{y} &> \tilde{n} \\ \tilde{x} + \tilde{z} &> \tilde{n} \\ \tilde{y} + \tilde{z} &> \tilde{n} \end{aligned}$$

Now we have a system of 4 linear equations with 3 variables, (recall that $\tilde{n}$ is a constant). The solution space for this system of equations is a tetrahedron with the following parameterization:

$$\begin{array}{rcccl} \tilde{n} - \tilde{y} & \leq & \tilde{z} & \leq & 2\tilde{n} - \tilde{x} - \tilde{y} \\ \tilde{n} - \tilde{x} & \leq & \tilde{y} & \leq & \tilde{x} \\ \tilde{n}/2 & \leq & \tilde{x} & \leq & \tilde{n} \end{array}$$

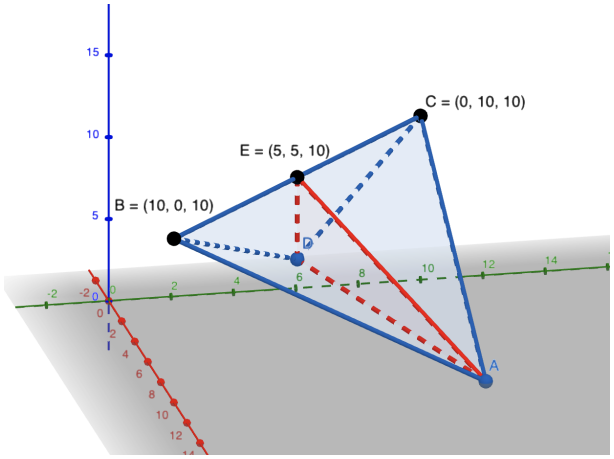


Figure 2: The tetrahedron solution space in log-log-log space with $n = 10$.

The density of primes in the new triple log space is now just $\frac{1}{\tilde{x}\tilde{y}\tilde{z}}$. We also have to use a Jacobian transformation to take into account the variable change.

$$\begin{vmatrix} \frac{\partial x}{\partial \tilde{x}} & \frac{\partial x}{\partial \tilde{y}} & \frac{\partial x}{\partial \tilde{z}} \\ \frac{\partial y}{\partial \tilde{x}} & \frac{\partial y}{\partial \tilde{y}} & \frac{\partial y}{\partial \tilde{z}} \\ \frac{\partial z}{\partial \tilde{x}} & \frac{\partial z}{\partial \tilde{y}} & \frac{\partial z}{\partial \tilde{z}} \end{vmatrix} = \begin{vmatrix} e^{\tilde{x}} & 0 & 0 \\ 0 & e^{\tilde{y}} & 0 \\ 0 & 0 & e^{\tilde{z}} \end{vmatrix} = e^{\tilde{x}+\tilde{y}+\tilde{z}}$$

Putting all these pieces, we get the following integral:

$$\psi(n) := \int_{\tilde{n}/2}^{\tilde{n}-1} \int_{\tilde{n}-\tilde{x}}^{\tilde{x}} \int_{\tilde{n}-\tilde{y}}^{2\tilde{n}-\tilde{x}-\tilde{y}} \frac{e^{\tilde{x}+\tilde{y}+\tilde{z}}}{\tilde{x}\tilde{y}\tilde{z}} \, d\tilde{x} \, d\tilde{y} \, d\tilde{z}$$

As of yet, we do not have a solution for this integral, so we will denote the integral by $\psi(n)$

Theoretically, we could also extend our count to include the product of 4 primes such that each partition has an element larger than n, or even 5 primes and so on, however, the process is similar to the above, but more tedious, and so we move on.

Using all these counts of sets to not include, we get an underestimate for the number of product nodes in $G(n)$ as:

$$n^2 - 1 \quad - \quad \frac{n}{\ln n} \quad - \quad n^2(\ln 2) \quad - \quad \psi(n)$$

7 Distributions

Next, we consider the distribution of degrees in both the sum and product nodes in a graph of $G(n)$

7.1 Distribution of Sum Node Degrees

As usual, analyzing the sum nodes is fairly simple. We have an equation in section 4.2 that details the degree of any given sum node, and using that equation it is simple to see that for any n value, and selected degree d, there are only 4 possible choices of sum nodes that would have a specific degree d. For example, the 4 sum nodes of degree 1 are the nodes 4, 5, $2n-1$ and $2n$. Similarly the 4 nodes of degree 2 are 6, 7, $2n-3$ and $2n-2$, and so on. Depending on the size of n, the total number of sum nodes may not be a multiple of 4, so there may not be exactly 4 nodes of the largest degree size, but that is the only exception.

Thus the distribution of sum node degrees takes on a uniform distribution where the frequency of each degree is exactly 4.

7.2 Distribution of Product Node Degrees

Again, this distribution is more complicated than the sum nodes. The way we went about this was to use a method similar to a Markov chain. Consider a graph $G(n)$ and a specific product node V_p with degree d. Then in the graph of $G(n+1)$, the degree of V_p will either remain d, or it will have degree $d+1$ if V_p is divisible by $n+1$.

The goal is to find the probability that a product node of degree 1 becomes a node of degree 2 as n increases by 1, we will call this probability β_1. Similarly, we want to find the probability that a product node of degree 2 becomes a product node of degree 3, we denote this probability β_2. We could also find the probabilities for nodes of larger degrees, but for now, we simply wish to find the distribution of degrees 1, 2 and 3. Lastly, because our system is not closed, and new nodes are introduced as n increases, we also need to find the number of new nodes of degree 1 that are added to the graph as n increases by 1, we denote

this number by α. One we have these three numbers, we can start with a distribution of degrees in $G(n)$, and calculate the distribution in $G(n+1)$. And by reiterating this process, we can find the distribution as n gets infinitely large.

We decided to use $n = 50$ as a base case. In $G(50)$ we have the following degree frequency:

$$\begin{bmatrix} |V_{p,d=1}(50)| \\ |V_{p,d=2}(50)| \\ |V_{p,d\geq 3}(50)| \end{bmatrix} = \begin{bmatrix} 518 \\ 162 \\ 104 \end{bmatrix}$$

Then to find the frequency of degrees at $n = 51$ we complete the following calculation:

$$\begin{bmatrix} 518 & 162 & 104 \end{bmatrix} \cdot \begin{bmatrix} 1-\beta_1 & \beta_1 & 0 \\ 0 & 1-\beta_2 & \beta_2 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} \alpha & 0 & 0 \end{bmatrix}$$

It is important to note that our variables β_1, β_2, α are all functions of n, so to reiterate this calculation, we cannot merely raise our matrix to higher powers. Instead, at the current moment in time, we are stuck merely re-doing the calculation for every time n increases by one.

So now that we know what we wish to accomplish, we must now calculate the values of these variables.

We start with the value of α. As a reminder, α is the number of new nodes added as n increases by 1. Another way of thinking about this, is that α is the number of product nodes of degree 0 that became degree 1. Fortunately, we have a function that estimates the total number of product nodes in $G(n)$, which we can denote $|V_p(n)|$. So we can estimate the value of α with the following calculation:

$$\alpha = |V_p(n)| - |V_p(n-1)|$$

Next, we will tackle β_2. The probability that a product node increases in its degree is equivalent to the probability that a node is divisible by n (not counting the value of n itself, as $1 * n$ is not a valid product). There are n-1 valid multiples of n between 4 and n^2, and we have to consider that approximately α of them are new nodes rather than existing nodes. So the probability that a node of degree 2 increases in it's degree from $G(n-1)$ to $G(n)$ is approximately:

$$\beta_2 = \frac{n-1-\alpha}{|V_p(n-1)|}$$

Lastly, we want the probability β_1. This is very similar to β_2, the only change we make is

that we acknowledge the fact that a lot of product nodes of degree 1 are fixed, as in their degree will not increase as n increases. This is specifically in the case of nodes which are a product of two prime numbers. So to calculate this probability, we use the same formula for β_2 but then also multiply it by the probability that a number is not the product of 2 primes. The number of numbers that are a product of two primes is $\binom{\pi(n)+1}{2}$, the probability that a number is the product of two primes is then $\frac{\binom{\pi(n)+1}{2}}{|V_p(n-1)|}$, and the probability that a number is *not* the product of two primes is one minus the above expression. Putting all these pieces together we get that

$$\beta_1 = \frac{n-1-\alpha}{|V_p(n-1)|} * \left(1 - \frac{\binom{\pi(n)+1}{2}}{|V_p(n-1)|}\right)$$

Now that we have expressions for the values of α, β_2 and β_2, we can plug them into our matrix, and using our base case of $n = 50$ we can reiterate the function and use a program to see how the distribution changes as n increases. We see that the distribution levels off pretty quickly and stabilizes at the following distribution

$$\begin{bmatrix} |V_{p,d=1}|/|V_p| \\ |V_{p,d=2}|/|V_p| \\ |V_{p,d\geq 3}|/|V_p| \end{bmatrix} = \begin{bmatrix} 0.58 \\ 0.20 \\ 0.22 \end{bmatrix} \approx \begin{bmatrix} 0.573 \\ 0.203 \\ 0.224 \end{bmatrix}$$

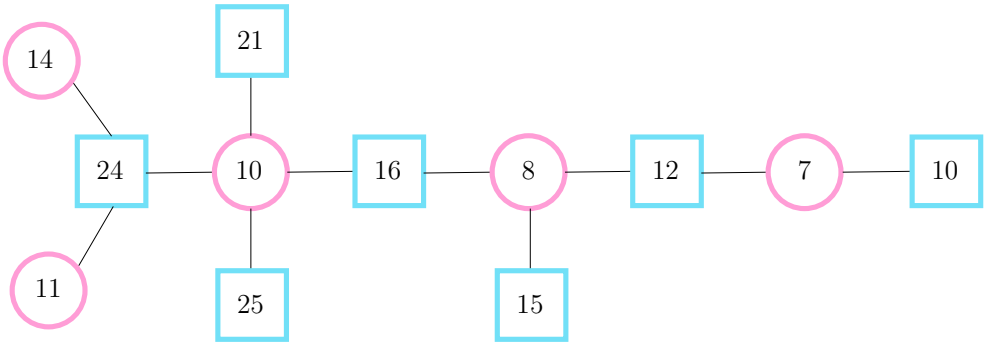
where the vector $\begin{bmatrix} 0.58 & 0.20 & 0.22 \end{bmatrix}^T$ is the estimate we get from our Markov chain, and the vector to the right of that $\begin{bmatrix} 0.573 & 0.203 & 0.224 \end{bmatrix}^T$ is the distribution generated by our code for large graphs. So it seems that our model does a pretty good job with the distribution estimate.

8 Probability Proof

The sum and product game is essentially a conversation between two players with different bits of information, and the game ends when either individual is able to discern the values of l and k. We know that the game can end after only a single turn, if the sum value can only be partitioned in one way. From data collection, we also know that the conversation can last up to 15 turns of each player not knowing the value of the sum and product before one person finally gets it. So the question we decided to ask is, for any game of size n, what is the longest length of the game, which is equivalent to finding the maximum chain length in a graph. We collected data for the longest chain length up until $n = 1000$ and found the following properties:

1. Almost half of the graphs had a maximum chain length of 6.
2. For the graphs that did not have maximum chain length of 6, their chain length was almost always an odd length.
3. Other than the graphs that had a chain length of 6, there are only 3 graphs with an even max chain length for all n values up to 1000.

First we aimed to understand the frequency of the length 6 chain. Upon closer inspection we were quickly able to detect that the 6-chain in all of these graphs is in fact the same chain, and it is the only permanent chain in all our graphs.



This chain starting with the product node 24 is found in every graph regardless of the size of n because the l and k values associated with these edges are all small, and the nodes do not have any further factorizations. So increasing n will not result in any additional edges being added to these nodes, which is why this chain is permanent. My research partner Nila proved that this chain is in fact the only permanent chain, and that any other would eventually connect to the rest of the graph and become a loop as n increases. The frequency of the value 6 as the maximum chain length is then easy to explain, as those are all the times this permanent chain is longer than any other chain in the graph.

The next thing we sought to understand is why the rest of the n values were much more likely to have an odd chain length over an even one. To do this we first analyze what it means to have an odd or even chain length graphically.
An odd chain length means that the chain starts with a sum node. Similarly, an even chain length must start with a product node. So our question now boils down to: why is it that a chain is more likely to start with a sum node instead of a product node?
The beginning of a chain must have the following properties:

- The node must be on the *edge* of the graph ($V_s(n) \geq 2n - k\sqrt{n}$).
- The degree of the node is at least 3.
- At least 3 neighbors of the node must have degree at least 2.

The first property ensures that we are looking at the significant areas of the graph, away from all the loops. We define the edge as the parts of the graph where the value of the sum nodes are greater than $2n - k\sqrt{n}$, where k is a constant that simply adjusts just how far on the edge we are. The second property is important to ensure that the node we are looking at has one edge that connects to the rest of the chain, and two other edges are a part of a loop. The third property ensures that the three neighboring nodes we just noted are in fact either connected to a chain or are part of a loop, and are not just leaves that dont have any other connections. With these requirements for a node to start a chain, we use our countings and

estimates to find the probability that a product node or a sum node meets these necessary requirements.
We begin by first defining some constants we will need for this probability.

- α = average degree of sum nodes on edge.
- p_1 = probability that product node on the edge has degree at least 2.
- p_2 = probability that product node on the edge has degree at least 3.

Once we have these values, we use a Bernoulli distribution to find the probability that a sum or product node is the start of a chain.
Sum: The probability that a sum node is on the edge of the graph, has a degree of at least 3, and is connected to at least 3 product nodes each of degree at least 2 is equal to

$$1 - bin(\alpha, 0, p_1) - bin(\alpha, 1, p_1) - bin(\alpha, 2, p_1). \tag{8.1}$$

Another way to think about this probability is that we find the probability that a sum node is connected to no product nodes of degree at least 2, or only 1 product node of degree at least 2, or only 2 product nodes of degree at least 2. Then we do one minus that value, which gives us the desired probability.

For a product, that last condition that it should be connected to at least 3 sum nodes of degree at least 2 is almost negligible. The reason for this being that out of all the sum nodes, only 4 of them have degree 1, so the probability that the neighboring sum nodes have degree at least 2 is can be approximated as 1. Therefore we do not bother taking that into account in our probability function.
Product: The probability that a product node is on the edge of the graph, has a degree of at least 3, and is connected to at least 3 sum nodes each of degree at least 2 is equal to

$$1 - bin(\alpha, 0, p_2) - bin(\alpha, 0, p_1). \tag{8.2}$$

Now all that is left is to find the values of α, p_1, p_2. The only problem is that all the distributions and countings that we have developed are for an average sum or product node, but we wish to focus on nodes on the edge of the graph, and their degree distribution is very different. So for now, we use our program to give us the degree distribution we need for the values of p_1 and p_2.

Figure 3 shows the distribution of degrees on the edge (where we let $k = 5$)for n ranging between 10 and 200.

Using the probabilities in the data above, we use equations 8.1 and 8.2 to find the probability that a product or sum node has the necessary conditions to begin a chain, below is our results in figure 4.

We see that the probability of a sum node starting a chain is approximately 0.45, whereas a product node lands at a probability of 0.075. This confirms our data which found it much more likely that a chain will be of odd length than even length, as desired.

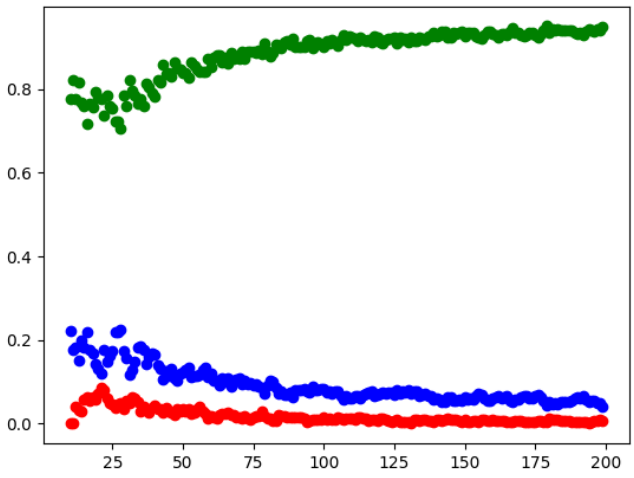


Figure 3: A distribution of the degrees for $k = 5$, and $10 \leq n \leq 200$.

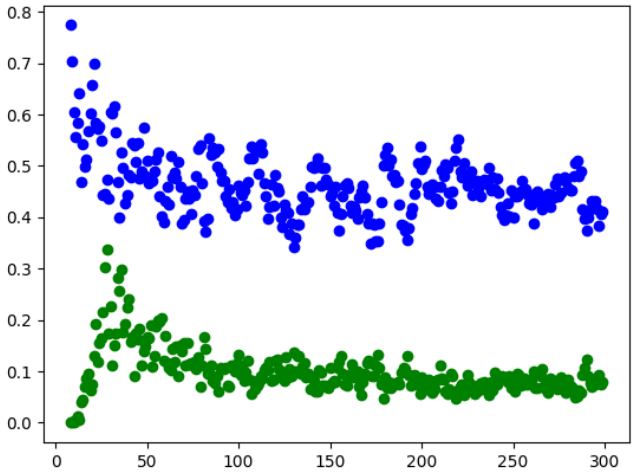


Figure 4: Probability outputted by equations 8.1 and 8.2. for $10 \leq n \leq 300$.
Blue: 8.1, Green: 8.2

9 Conclusion and Future Research

Throughout this paper, we have managed to develop countings and estimates for various features of the graph that aided us in the beginning of a probability proof regarding the patterns behind the parity of chain lengths. However, most of our countings were of the entire graph, and when developing our proof, we found that it was necessary to address the features of nodes in more specific regions of the graph, specifically those on the edge (away from the loops in the middle). Future research would aim to adjust our countings

and estimates to be able to focus on specific regions of the graph. We would also attempt to improve our estimate on the total number of product nodes, as our current estimate is not terribly accurate for n values larger than 1000. In addition to tightening our current estimate, which serves as a lower bound, we would also attempt to create an upper bound as well for the total number of product nodes.

10 Appendix: Code

```
from turtle import color
import networkx as nx
import itertools as it
import matplotlib.pyplot as plt
from math import*
import sympy
import numpy as np
from sklearn.linear_model import LinearRegression
import re
import collections
import pandas as pd
from random import randint
from tabulate import tabulate
from matplotlib.pyplot import cm
import scipy.stats as ss
from scipy import exp
from scipy.integrate import tplquad


##############################################################################################
# function that prints graph
##############################################################################################


def thegraph(MAX):
    # Building the graph
    G = nx.Graph()
    for i, j in it.combinations_with_replacement(range(2, MAX+1), 2):
        sum_node = f'S{i+j}' # 'S6' for i=2, j=3
        prod_node = f'P{i*j}'
        G.add_edge(sum_node, prod_node)

    color_map = ['pink' if node.startswith('S') else 'lightblue' for node in G]
    options = {
        'node_color': color_map,
        'node_size': 600,
        'font_size': 10,
        'width': 0.8,
        'with_labels': True,
    }

    plt.figure(1, figsize=(16,6)) # Add 1, 2,.. for additional figures
    nx.draw(G, nx.nx_agraph.graphviz_layout(G), **options)

    # Removing the leaves
```

```
    leaves = [node for node in G if G.degree(node) <= 1]
    G.remove_nodes_from(leaves)

    # Drawing the graph without leaves
    color_map = ['pink' if node.startswith('S') else 'lightblue' for node in G]
    options['node_color'] = color_map

    plt.figure(2, figsize=(10,6))
    nx.draw(G, nx.nx_agraph.graphviz_layout(G), **options)

    # leaves left after 4 levels of removal
    i = 2
    for i in range(2, 5):
        leaves = [node for node in G if G.degree(node) <= 1]
        G.remove_nodes_from(leaves)
        i += 1

    plt.show()



##############################################################################################
#function that gives you the various partitions of a sum
##############################################################################################

def sums(n, numb):
    answer = set()
    p1 = floor(numb/2)
    p2 = ceil(numb/2)
    while p1 >= 2 and p1 <= n:
           answer.add(tuple((p1, p2)))
           p1 = p1-1
           p2 = p2+1
    return answer


##############################################################################################
# printing longest chain (defining functions)
##############################################################################################

# Building the graph
def BuildGraph(n):
  G = nx.Graph()
  for i, j in it.combinations_with_replacement(range(2, n+1), 2):
    edge = (f'S{i+j}', f'P{i*j}')
    G.add_edge(*edge, component=(i, j))
  nx.set_node_attributes(G, {node: [] for node in G}, 'chain')
```

```
  return G

# Removing the leaves
def RemoveLeaves(G):
  leaves = [node for node in G if G.degree(node) == 1]
  for leaf in leaves:
    neighbor = list(G.neighbors(leaf))[0]
    long_chain = G.nodes[leaf]['chain'] + [leaf]
    if len(G.nodes[neighbor]['chain']) < len(long_chain):
      nx.set_node_attributes(G, {neighbor: long_chain}, 'chain')
  G.remove_nodes_from(leaves)
  return leaves

def LongestPath(G):
  for i in it.count():
    if not RemoveLeaves(G):
      break
  return max([G.nodes[node]['chain']+[node] for node in G.nodes], key=len)


##############################################################################
# longest chain length
##############################################################################

def max_chain(MIN, MAX):
  for n in range(MIN, MAX + 1):
    # Building the graph
    G = nx.Graph()
    for i, j in it.combinations_with_replacement(range(2, n+1), 2):
        sum_node = f'S{i+j}' # 'S6' for i=2, j=3
        prod_node = f'P{i*j}'
        G.add_edge(sum_node, prod_node)

    # finding the longest chain length

    chain = LongestPath(BuildGraph(n))
    leaves = [node for node in G if G.degree(node) <= 1]
    j = -1
    while len(leaves) > 0:
      leaves = [node for node in G if G.degree(node) <= 1]
      G.remove_nodes_from(leaves)
      j += 1
      if len(leaves) == 0:
        print(j)
        print()
```

```
##########################################################################################
#Preliminary counting functions
##########################################################################################


def combi(n, r):
  return factorial(n)/(factorial(n-r)*factorial(r))

def rep_combi(n, r):
  return combi(n+r-1, r)

def pi2(x):
  #return sympy.primepi(x)
  return floor(x/log(x))


##########################################################################################
# number of product nodes
##########################################################################################

def ProductNodes(n):
  G = BuildGraph(n)
  noodles = list(G.nodes())
  pewpew = len([x for x in noodles if x.startswith('P')])
  return(pewpew)


##########################################################################################
#Product Nodes Estimation Functions
##########################################################################################

def estim_prod_node_10(n):
  p1 = n**2*(log(2)) #number of primes and their multiples
  p2 = pi2(n) # primes smaller than n
  func = lambda z, y, x: exp(x+y+z)/(x*y*z)
  N = log(n)
  x1, x2 = N/2, N-1
  y1, y2 = lambda x: N-x, lambda x: x
  z1, z2 = lambda x, y: N-y, lambda x, y: 2*N-x-y
  p3 = tplquad(func, x1, x2, y1, y2, z1, z2)
  return n**2 - p1 -p2 - p3[0]

##########################################################################################
#Probability functions
##########################################################################################
def bin_dist(n, r, p):
  return combi(n, r)* p**r *(1-p)**(n-r)
```

```
def probability_sum2(n):
  k = 5
  p1 = 1 - final(n)[0]                    #prob a prod node has degree 2 or more
  a = floor(s_avg(n))

  return 1-(bin_dist(a, 0, p1) + bin_dist(a, 1, p1)
  + bin_dist(a, 2, p1))


def probability_prod2(n):
  p1 = 1- final(n)[0]
  k = 5
  p2 = 1- final(n)[0] - final(n)[1] #prob a prod node has degree 3 or more
  a = floor(s_avg(n))
  return (1- (bin_dist(a, 0, p2) - bin_dist(a, 0, p1)))
```

References

[1] M. Gardner. Mathematical games. *Scientific American*, 241(6):22–32, 1979.

[2] G. SHAKAN. On higher energy decompositions and the sum–product phenomenon. *Mathematical Proceedings of the Cambridge Philosophical Society*, 167(3):599–617, jul 2018.