A Statistical Look at Maps of the Discrete Logarithm

Nathan Lindle

Rose-Hulman Institute of Technology

Follow this and additional works at: https://scholar.rose-hulman.edu/math_mstr

Recommended Citation

https://scholar.rose-hulman.edu/math_mstr/35

This Article is brought to you for free and open access by the Mathematics at Rose-Hulman Scholar. It has been accepted for inclusion in Mathematical Sciences Technical Reports (MSTR) by an authorized administrator of Rose-Hulman Scholar. For more information, please contact weir1@rose-hulman.edu.
A Statistical Look at Maps of the Discrete Logarithm

Nathan Lindle

Adviser: Joshua Holden

Mathematical Sciences Technical Report Series
MSTR 08-06

May 21, 2008

Department of Mathematics
Rose-Hulman Institute of Technology
http://www.rose-hulman.edu/math
1 Introduction

Cryptography is being used today more than it ever has in the past. Millions of transactions are being conducted every hour using encrypted channels, most of which use the Internet as their medium. It is taken for granted by the average user that these transactions are secure, but mathematicians and computer scientists alike are constantly testing the algorithms being used. Several of these cryptosystems use the transformation

\[ g^x \equiv y \pmod{n} \]  

The appeal of this transformation is that it is quite simple to calculate \( g^x \bmod{n} \); exponentiation by squaring is fairly simple and quick even using very large numbers. However there is no known algorithm to compute the inverse of the transformation (that is given \( y, g, \) and \( n \), find \( x \)) in a comparable amount of time. This is called the discrete log problem, and Diffie-Hellman key exchange, RSA and the Blum-Micali pseudorandom bit generator all rely on its inherent difficulty. This paper explores the maps generated by this transformation with the hope of gaining some insight into its structure and similarity to random mappings.

2 Functional Graphs

2.1 Terminology

A directed functional graph is a set of nodes \( S \) and a set of directed edges where \((a, b)\) denotes an edge from node \( a \) to node \( b \) on the graph. Also, every node must have exactly one edge coming out of it (this is where the “functional” part of the name comes from). Using the transformation from (1), define a transition function \( \varphi \) where \( \varphi(x) = g^x \pmod{p} \). Given any prime modulus \( p \) and some base \( g \), construct a functional graph using \( \varphi \) where \( S = \{1, 2, \ldots, p-1\} \) and the directed edges are \((a, \varphi(a))\) for every \( a \in S \).

Now take any random starting node \( u_0 \), construct the sequence \( u_1 = \varphi(u_0), u_2 = \varphi(u_1), \ldots \), and note that there are \( p-1 \) nodes in the graph. After at most \( p \) iterations, it must be the case that \( u_i = u_j \) where \( i < j \). \( j \) is called the rho length of \( u_0 \) (because when you graph the sequence it looks like a \( \rho \) ).
is the tail length of $u_0$ and $j - i$ is the cycle length of $u_0$. Note that it is possible for $u_0$ to be part of a cycle, so the tail length could be 0. For example, nodes 3 and 8 in Figure 1 have a cycle length of 1, and the rest of the nodes have a cycle length of 3. Nodes 2 and 7 both have a tail length of 2, and nodes 10, 6, 4 and 8 have tail length 1, and the rest have a length of 0. The average tail length of a graph is the sum of the tail lengths of each node divided by the number of nodes. Similarly, the average cycle length is the sum of the cycle lengths of each node divided by the number of nodes. Figure 1 has an average tail length of 0.8 and an average cycle length of 2.6, for instance.

A node $x$ is an image node if there is a transition $(a, x)$ for some $a \in S$. Otherwise $x$ is called a terminal node. A cycle and all the nodes included in tails which lead to that cycle is called a connected component. Since the out-degree of every node is exactly one, no component is connected to any other, and each component has exactly 1 cycle.

The maximum cycle length for a graph is the size of the largest cycle (which may or may not be a part of the largest component). The maximum tail length is the maximum number of transitions any node is from its cycle, or equivalently the maximum tail length of any node.

The in-degree of a node $b$ is the number of edges of the form $(a, b)$ for some node $a$ in the graph. A functional graph is said to be $m$-ary if the in-degree of each node is either 0 or $m$. 1-ary graphs are referred to as permutations, and 2-ary graphs are referred to as binary functional graphs. Therefore the graph in Figure 1 can be more specifically labeled as a binary functional graph.

### 2.2 Previous work

Daniel Cloutier did a lot of work analyzing functional graphs using this transition function in [2]. Theorem 1 from his paper proves an important result needed to practically analyze these graphs, and it is
restated here:

**Theorem 1 (Theorem 1 of [2])** Let \( p \) be fixed and let \( m \) be any positive integer that divides \( p - 1 \). Then as \( g \) ranges over all integers, there are \( \phi\left(\frac{p-1}{m}\right) \) different functional graphs which are \( m \)-ary produced by maps of the form \( f : x \mapsto g^x \mod p \). Furthermore, if \( r \) is any primitive root modulo \( p \), and \( g \equiv r^a \mod p \), then the values of \( g \) that produce an \( m \)-ary graph are precisely those for which \( \gcd(a, p-1) = m \).

This theorem allows a person to not only know how many \( m \)-ary graphs can be generated from a given prime \( p \), but allows them to figure out which bases (values of \( g \)) will generate such graphs. A generator \( r \) is an element of the residue class of \( p \) such that any element \( g \) can be written as \( r^a \equiv g \mod p \). One only needs to pick a generator and do a simple check on each possible \( a \) to see if \( \gcd(a, p - 1) = m \), and use \( r^a \) as the base if this is the case.

This result should also make it evident that it makes sense to separate our analysis of the graphs based on what type of \( m \)-ary graph it is. The author of [2] showed that this is indeed the case; considering all possible graphs for a given \( p \) does not produce results similar to a random functional graph, even if there are many possible types of graph (a result that arises when \( p - 1 \) is very composite). When permutations and binary functional graphs, which correspond to \( m = 1 \) and \( m = 2 \), were considered separately however, a more satisfying analysis could be performed.

The author of [2] extended a technique from [5] which uses generating functions to determine the expected values for some parameters of interest in random binary functional graphs. The techniques used will be explored more in depth in the Theoretical Values section, but the parameters analyzed in each graph were number of components, number of cyclic nodes, number of tail nodes, number of terminal nodes, average cycle length, average tail length, maximum cycle length and maximum tail length. For a given prime \( p \), the author of [2] generated as many binary functional graphs as possible and collected data on these parameters for each one. Since each graph had the same number of nodes, these results could then be combined and compared to the theoretical parameter values random binary functional graphs have. The results for the 3 primes he collected data on indicated that these graphs indeed had a similar structure to random binary functional graphs.

The work on binary functional graphs in [2] has been extended further to take an even closer look at the graphs generated. Principally, this paper seeks to analyze the variance in each of the parameters mentioned before, as well as the maximum tail length, and also seeks an exact value for many of the theoretical parameters instead of an asymptotic one. Many more tests have also been run over a much wider range of primes in order to increase the confidence in the accuracy of the results.

### 3 Theoretical Values

#### 3.1 Techniques

The first step to creating the generating functions needed is to describe the structure of the graphs, as in [5]. A binary functional graph will be a set of components, each of which is made up of a cycle of nodes with a binary tree attached to each node. A binary tree is a node with up to 2 binary trees attached to it. Converting this English description to the same notation as [5] yields

\[
\begin{align*}
\text{BinFunGraph} & = \text{set(Components)} \\
\text{Component} & = \text{cycle(Node*BinaryTree)} \\
\text{BinaryTree} & = \text{Node + Node*set(BinaryTree, cardinality = 2)} \\
\text{Node} & = \text{Atomic Unit}
\end{align*}
\]
Using the techniques of [5], this can then be converted to the following generating functions:

\[ f(z) = e^{c(z)} = \frac{1}{1 - zb(z)} \]  

(2)

\[ c(z) = \log \frac{1}{1 - zb(z)} \]  

(3)

\[ b(z) = z + \frac{1}{2} zb^2(z) \]  

(4)

\( f \) here generates functional graphs, so the coefficient of \( z^n \) in the Taylor expansion of \( f(z) \) will be \( \frac{f_n}{n!} \) where \( f_n \) is the number of binary functional graphs of size \( n \). Similarly, the coefficients of \( z^n \) in \( c(z) \) and \( b(z) \) will tell us about the number of components and the number of binary trees. Solving the quadratic equation for \( b(z) \), and using the only answer that makes sense, gives a simpler representation of \( f \) and \( c \):

\[ f^*(z) = \frac{1}{\sqrt{1 - 2z^2}} \]  

(5)

\[ c^*(z) = \log \frac{1}{\sqrt{1 - 2z^2}} \]  

(6)

In [2], the author uses singularity analysis on these generating functions to get an asymptotic form for the coefficients. This paper takes a different route, seeking to solve for the coefficient directly where possible. The first step is to create a differential equation \( y(z) \) with a set of initial conditions which is satisfied by the function. Applying this process to \( f^* \) yields

\[ y(0) = 1 \]

\[ 2zy(z) + (-1 + 2z^2) \left( \frac{d}{dz} y(z) \right) = 0 \]

One can then find the power series solution to this differential equation and use it to create the recurrence \( u(n) \), which will give the \( n \)th Taylor coefficient for the generating function around 0. The initial condition for the differential equation is used to create the base case for the recurrence. In this case, solving for this recurrence gives

\[ u(0) = 1 \]

\[ u(1) = 0 \]

\[ (2n + 2)u(n) - (n + 2)u(n + 2) = 0 \]

After studying this result briefly, one can see that \( u(2n + 1) \) will be 0 for any positive \( n \). This follows directly from the description of binary functional graphs, since such a graph cannot be created with an odd number of nodes. Both the conversion to a differential equation and the conversion to a recurrence relation were done using functions from [6]; more documentation can be found there. Solving this recurrence yields

\[ g(n) = \frac{2^{\frac{n}{2}} \Gamma(n/2 + 1/2)}{\sqrt{\pi} \Gamma(n/2 + 1)} \]  

(7)

where \( \Gamma(n) = \int_{t=0}^{\infty} e^{-t^n} \). \( g(n) \) therefore will be the number of possible binary functional graphs of size \( n \) divided by \( n! \). In the following sections, a similar technique will be used to get functions for parameters of interest.
3.2 Means

**Theorem 2** The expected number of components, number of cyclic nodes, average cycle length and average tail length in a random binary functional graph of size $n$ are

Number of Components
$$\left(\frac{1}{2}\right)\Psi\left(\frac{n}{2} + \frac{1}{2}\right) + \left(\frac{1}{2}\right)\gamma + \ln(2) \quad (8)$$

Number of Cyclic Nodes
$$\frac{\sqrt{n}\Gamma\left(\frac{n}{2} + 1\right) - \Gamma\left(\frac{n}{2} + \frac{1}{2}\right)}{\Gamma\left(\frac{n}{2} + \frac{1}{2}\right)} \quad (9)$$

Average Cycle Length
$$\frac{\frac{1}{2}\sqrt{n}\Gamma\left(\frac{n}{2} + 1\right)}{\Gamma\left(\frac{n}{2} + \frac{1}{2}\right)} \quad (10)$$

Average Tail Length
$$\frac{\sqrt{n}\Gamma\left(2 + \frac{n}{2}\right) - n\Gamma\left(\frac{n}{2} + \frac{1}{2}\right) - \Gamma\left(\frac{n}{2} + \frac{1}{2}\right)}{n\Gamma\left(\frac{n}{2} + \frac{1}{2}\right)} \quad (11)$$

Here, $\Psi(n) = \left(\frac{d}{dx}\Gamma(x)\right)/\Gamma(x)$ and $\gamma$ represents the Euler constant, which is approximately 0.57721566.

**Proof.** The generating functions for these parameters were developed in [2], and in following with the notation from [5], they are called $\Xi(z)$. Since the equations can get quite large, only the process for computing the average cycle length is reproduced here. The other functions are created in exactly the same manner, but can get quite messy. The generating function $\Xi(z)$ for the average tail length is:

$$\Xi(z) = \frac{2z^2(1 - \sqrt{1 - 2z^2})}{(1 - 2z^2)^{3/2}} + \frac{2z^2}{(1 - 2z^2)^{3/2}} \quad (12)$$

Using the same techniques as earlier, a differential equation can be found that is satisfied by $\Xi(z)$, which can then be used to get the following recurrence:

$$4u(n) - 4u(n + 2) + u(n + 4) \quad (13)$$

$$u(1) = 0 \quad u(0) = 0 \quad u(2) = 2 \quad u(3) = 0$$

This recurrence can then be solved to give the surprisingly simple solution of

$$[z^n]\Xi(z) = 2^{n/2}(n/2) \quad (14)$$

This is of course only true when $n$ is even. To find the expected value for the cycle length for a given graph, this result needs to be normalized. First it must be multiplied by $n!$, since $c$ is the actual parameter of interest and $[z^n]\Xi(z) = c/n!$. The result should also be divided by the total number of binary functional graphs (to get an expected value), and finally by $n$ because this parameter (as well as average tail) is taking results as seen from a random node in the functional graph. This final division is not performed for the calculation of the expected number of components and cyclic nodes. Since $g(n)$
actually equals the number of binary functional graphs divided by \( n! \), the result for Theorem 2 can be obtained by expanding

\[
\frac{[z^n]\Xi(z)}{n(g(n))}
\]

Unfortunately the techniques to calculate maximum tail length and maximum cycle length expected values uses other approximations which prevent a direct recursive calculation at this time. However, since much more data has been collected for this paper, it will be interesting to look at the previously determined asymptotic estimations of these parameters. They are reproduced here for reference:

**Theorem 3 (Theorem 7 of [3])** The asymptotic forms for the expected sizes of the largest cycle and the largest tail in a random binary functional graph of size \( n \), as \( n \to \infty \), are

Largest Cycle

\[
\sqrt{\frac{\pi n}{2}} \int_0^\infty \left[ 1 - \exp \left(- \int_v^\infty e^{-u \frac{du}{u}} \right) \right] \frac{dv}{\sqrt{v}} \approx 0.78248 \sqrt{n}
\]

Largest Tail

\[
\sqrt{2\pi n \ln(2) - 3 + 2 \ln(2)} \approx 1.73746 \sqrt{n} + 1.61371
\]

### 3.3 Variances

**Theorem 4** The variance in number of components, number of cyclic nodes, average cycle length and average tail length between binary functional graphs is

Number of Components

\[
\frac{1}{2} \Psi \left( \frac{n}{2} + \frac{1}{2} \right) + \left( \frac{1}{2} \right) \gamma + \ln(2) + \frac{1}{4} \gamma^2 + \gamma \ln(2) + \ln(2)^2 + \sum_{i=0}^{n/2-1} \frac{\Psi(l + \frac{1}{2})}{(2l + 1)} - \frac{1}{4} \Psi \left( \frac{n}{2} + \frac{1}{2} \right)^2
\]

Number of Cyclic Nodes

\[
-\frac{(-2\Gamma(\frac{n}{2} + \frac{1}{2})^2 - 4\frac{\sqrt{\pi}}{\Gamma(\frac{n}{2} + \frac{1}{2})} \Gamma(\frac{n}{2} + \frac{1}{2}) \Gamma(\frac{n}{2} + \frac{1}{2}) + \pi \Gamma(\frac{n}{2} + 1)^2)}{\Gamma(\frac{n}{2} + \frac{1}{2})^2}
\]

Average Cycle Length

\[
\frac{1}{12} \left( -3\pi \Gamma(\frac{n}{2} + 1)^2 - 6\sqrt{\pi} \Gamma(\frac{n}{2} + 1) \Gamma(\frac{n}{2} + \frac{1}{2}) + 16\frac{\pi}{2} \Gamma(\frac{n}{2} + \frac{1}{2})^2 + 8\Gamma(\frac{n}{2} + \frac{1}{2})^2 \right)
\]

Average Tail Length

\[
\frac{1}{6} \left( -18\Gamma(\frac{n}{2} + \frac{3}{2}) - 8\Gamma(\frac{n}{2} + \frac{3}{2}) \frac{\sqrt{\pi}}{\Gamma(\frac{n}{2} + 1)} + 9\frac{\sqrt{\pi}}{\Gamma(\frac{n}{2} + 1)} + 9\sqrt{\pi} \Gamma(\frac{n}{2} + 1) \right)
\]

\[
\frac{1}{4} \left( \sqrt{\pi} \Gamma(\frac{n}{2} + 1) + \frac{\sqrt{\pi}}{\Gamma(\frac{n}{2} + 1)} - 2\Gamma(\frac{n}{2} + \frac{3}{2}) \right)^2
\]

**Proof.** As the reader can undoubtedly see, these equations can become quite large. For this reason, the proof will again be limited to determining the average cycle value. The variance for a set of data is

\[
\frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2
\]

where \( N \) is the number of data points, \( \bar{x} \) is the mean, and the \( x_i \) are the individual data points.
points. Using some simple algebra this is equivalent to \( \frac{1}{N} \left( \sum_{i=1}^{N} x_i^2 \right) - x^2 \), so a generating function for \( \left( \sum_{i=1}^{N} x_i^2 \right) \) would be very handy.

Using the techniques from [1], a doubly marked function for the average cycle length in a random binary functional graph is

\[
\xi(z) = \ln \left( \frac{1}{1-u(1-\sqrt{1-2z^2})} \right) \sqrt{1-2z^2} \tag{22}
\]

To compute \( \Xi \), which was used to find the expected average cycle length, the authors of [1] took \( \xi(z) = \frac{\partial}{\partial w} \xi(z) \bigg|_{w=1} \) to eventually obtain \( [z^n]\xi(z) \) (this was actually done for tertiary graphs, but the idea is the same). Since \( u \) is marking the parameter of interest, differentiating with respect to \( u \) then substituting \( u = 1 \) is like summing up the parameter for each graph of size \( n \) then putting it in front of \( z^n \). A technique to get \( \left( \sum_{i=1}^{N} x_i^2 \right) \) would be to differentiate with respect to \( u \), then multiply by \( u \) to correct for the power of \( u \), then differentiate with respect to \( u \) again. Now the coefficient of the \( u \) terms is the parameter squared, so substituting in \( u = 1 \) gives the needed generating function. Doing this with a doubly marked generating function does not introduce any problems, the \( w \) was only introduced to account for the fact that the average cycle length is a statistic taken from every node. Therefore the generating function for the total cycle length squared is

\[
\Xi^*(z) = \frac{d}{du} \left( u \left( \frac{\partial}{\partial w} \xi(z) \bigg|_{w=1} \right) \right) \bigg|_{u=1} \tag{23}
\]

which turns out to be

\[
\Xi^*(z) = -\frac{6z^2(-1 + \sqrt{1-2z^2})}{(1-2z^2)^2} + \frac{2z^2}{(1-2z^2)^{3/2}} + \frac{4z^2(-1 + \sqrt{1-2z^2})^2}{(1-2z^2)^{5/2}} \tag{24}
\]

Turning this into a differential equation, then a recurrence relation, and solving for the recurrence relation (as demonstrated previously in the paper) gives

\[
[z^n]\Xi^*(z) = \frac{1}{3} \left( \frac{3n}{2} \sqrt{\pi} \Gamma \left( \frac{n}{2} \right) - 8 \Gamma \left( \frac{n}{2} + \frac{3}{2} \right) \right) \sqrt{\pi} \Gamma \left( \frac{n}{2} \right) \tag{25}
\]

Now, let \( v(z) \) be the variance in the average cycle length of graphs of size \( z \). The mean has already been described as \( \frac{[z^n]\Xi(z)}{n\sigma(g(n))^2} \), so combining this result with the one just obtained, and using them with the simplified equation for variance computed at the beginning of the proof, yields

\[
v(z) = \frac{[z^n]\Xi^*(z) g(n) n - ([z^n]\Xi(z))^2}{n^2 g(n)^2} \tag{26}
\]

When expanded, \( v(z) \) is the same value as in the theorem. The rest of the variances in the parameters (components, cyclic nodes and average tail) are computed similarly with the only difference from the computation of the average cycle being in the normalization, just like the mean.
The key to the success of this technique is having a marked generating function. Since the generating functions for the maximum cycle length and the maximum tail length were not computed using marked generating functions, and no marked generating functions have been found yet, a theoretical value for the variance in these two statistics does not yet exist. Observed values for the variance were collected though in order to better compare the means to the expected values.

4 Observed Values

The same techniques as [2] were used to gather data on functional graphs which use (1) as the transition function. A large prime \( p \) is chosen, and a generator \( r \) is chosen for the integers modulo \( p \), then a graph is generated using \( g = r^a \) for each possible \( a \in \{1, 2, 3, 4, ..., p-1\} \) where \( \gcd(a, p-1) = 2 \). For each prime, the results are then combined to get the average number of components, cyclic nodes, cycle length, tail length, maximum cycle and maximum tail. Data for the variances of these parameters is collected as well. The code to generate these graphs is based on the C++ code written by the author of [2], but it has been converted to C, optimized in places, parallelized and altered to collect data for the computation of the variance.

The total number of graphs generated depends on \( \phi(p-1) \). The primes to test were chosen in three sets of 10. The first set is around 100,000, the second set is all safe primes (primes which can be written as \( 2q + 1 \) where \( q \) is prime) which are basically spread evenly between 110,000 and 200,000, and the last set is primes which are spread over the same interval and have a very composite \( p-1 \). For a list of the number of graphs generated for each prime, see Appendix A.

4.1 Means

For a given \( p, \phi((p-1)/2) \) graphs can be generated. In order to compare the average values of the parameters in these graphs with the values that Theorems 2 and 3 predict, this paper uses a t-test on the means. The idea here is to ascribe some statistical significance to the results, instead of just saying that they look similar. When performing a t-test, the experimenter has a hypothesis in mind which he would like to verify, \( H_0 \), and an alternative hypothesis \( H_a \). The t-test on the data produces a t-statistic, which can then be matched with a table to get a P-value. The P-value is the probability of obtaining a test statistic which is stranger, or more extreme, than the one that was observed, given that the null hypothesis is false. So low P-values indicate that the data collected may be unusual, and high P-values give no indication that the null hypothesis is true. In this case, an observed mean (for components for instance) \( \mu_{\text{obs}} \) is compared to a predicted mean \( \mu_{\text{pred}} \) (from Theorem 2 or 3). The null hypothesis is \( H_0 : \mu_{\text{pred}} = \mu_{\text{obs}} \) and the alternative hypothesis is \( H_a : \mu_{\text{pred}} \neq \mu_{\text{obs}} \). The t-test is

\[
\frac{(\mu_{\text{pred}} - \mu_{\text{obs}}) \sqrt{N}}{\sigma_{\text{obs}}}
\]

where \( N = p - 1 \) in this case and \( \sigma_{\text{obs}} \) is the standard deviation (which is the square root of the variance) in the observed samples. The number of degrees of freedom here is very large (\( df >> 1000 \)), so the t-statistic is actually the same as a z-statistic and we can use a z table to get the corresponding P-value. Since our null hypothesis allows for \( \mu_{\text{pred}} > \mu_{\text{obs}} \) or \( \mu_{\text{pred}} < \mu_{\text{obs}} \), the value obtained from the table is doubled to get the actual P-value (this is called a 2-tailed t-test). For more information on these tests see [4]. Table 1 gives an example of the results of this test for a single prime and its associated graphs.
<table>
<thead>
<tr>
<th>Components</th>
<th>Predicted</th>
<th>Observed</th>
<th>t-value</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Components</td>
<td>6.392</td>
<td>6.389</td>
<td>0.266</td>
<td>0.842</td>
</tr>
<tr>
<td>Cyclic Nodes</td>
<td>395.417</td>
<td>395.303</td>
<td>0.123</td>
<td>0.920</td>
</tr>
<tr>
<td>Average Cycle</td>
<td>198.208</td>
<td>198.319</td>
<td>-0.173</td>
<td>0.920</td>
</tr>
<tr>
<td>Average Tail</td>
<td>197.212</td>
<td>197.178</td>
<td>0.088</td>
<td>1.000</td>
</tr>
<tr>
<td>Maximum Cycle</td>
<td>247.495</td>
<td>247.261</td>
<td>0.339</td>
<td>0.764</td>
</tr>
<tr>
<td>Maximum Tail</td>
<td>547.935</td>
<td>541.827</td>
<td>8.359</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Table 1: Observed means, t-values and corresponding P-values for the prime 100043

All of the P-values obtained for the maximum tail statistic were very low, and in general the observed values are a bit lower than the theoretical means. Some other statistics have very low P-values for some primes, but have very high P-values for other primes. Based on the definition of a P-value, a uniform distribution is expected when looking at the same statistic over all of the \( p \) which have data. Therefore some low values are expected along with some high ones (it would actually be unusual to have P-values which are all very high). To tell if this is actually what’s going on for these P-values, one could perform some sort of uniformity test on them. Alternatively, one could perform a normality test (of which there are a large number to choose from) on the test statistics since the t-test uses so many degrees of freedom that it is like a z-test, which uses a normal curve. Table 2 shows the P-value obtained from an Anderson-Darling normality test on each statistic where the 33 t-test results were used as input.

<table>
<thead>
<tr>
<th></th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Components</td>
<td>0.632</td>
</tr>
<tr>
<td>Cyclic Nodes</td>
<td>0.069</td>
</tr>
<tr>
<td>Average Cycle</td>
<td>0.483</td>
</tr>
<tr>
<td>Average Tail</td>
<td>0.084</td>
</tr>
<tr>
<td>Maximum Cycle</td>
<td>0.102</td>
</tr>
</tbody>
</table>

Table 2: P-values obtained by doing Anderson-Darling normality tests on the t-statistics from the mean tests. The maximum tail length statistic is excluded because the P-values from the t-test were consistently very low. Tests were run using Minitab.

### 4.2 Variance

Techniques similar to those used to compare means will also be useful to take a statistical look at the observed variance compared to the predicted variance. Here the null hypothesis is \( H_0 \) : \( \sigma^2_{\text{obs}} = \sigma^2_{\text{pred}} \) and the alternative hypothesis is \( H_a \) : \( \sigma^2_{\text{obs}} \neq \sigma^2_{\text{pred}} \). The test used to get the P-value when comparing these is very similar to the one used to compare means; it is

\[
\frac{(\sigma^2_{\text{pred}} - \sigma^2_{\text{obs}})\sqrt{N}}{\tau_{\text{obs}}}
\]

where \( \tau_{\text{obs}} \) is used to denote \( \text{Var}(\{(x_i - \bar{x}_{\text{obs}})^2\}) \), the variance between the square of the difference between an individual data point (the number of components in a certain functional graph for instance) and the mean value for that statistic. This produces another statistic which can be used to find a P-value in a
z-table. Since no theoretical variances have been obtained for the maximum tail and maximum cycle, this test cannot be performed on those statistics. Table 3 gives an example of the results of this test on data from a single prime.

<table>
<thead>
<tr>
<th></th>
<th>Predicted</th>
<th>Observed</th>
<th>t-value</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Components</td>
<td>5.158</td>
<td>5.117</td>
<td>1.227</td>
<td>0.230</td>
</tr>
<tr>
<td>Cyclic Nodes</td>
<td>42543.192</td>
<td>42227.348</td>
<td>1.123</td>
<td>0.272</td>
</tr>
<tr>
<td>Average Cycle</td>
<td>27210.527</td>
<td>20392.727</td>
<td>35.112</td>
<td>0.000</td>
</tr>
<tr>
<td>Average Tail</td>
<td>27210.956</td>
<td>7362.882</td>
<td>302.252</td>
<td>0.000</td>
</tr>
<tr>
<td>Maximum Cycle</td>
<td></td>
<td>23806.218</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum Tail</td>
<td></td>
<td>26709.198</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Predicted and observed variances, t-values and corresponding P-values for the prime 100043. Theoretical values for the maximum cycle and maximum tail variances do not exist.

The observed variances in the average tail length and the average cycle length are quite a bit lower than expected. The P-values for the other statistics look like those obtained for the means, so it useful to perform a normality test on these test statistics as well. Table 4 shows the results from these tests.

<table>
<thead>
<tr>
<th></th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Components</td>
<td>0.485</td>
</tr>
<tr>
<td>Cyclic Nodes</td>
<td>0.487</td>
</tr>
</tbody>
</table>

Table 4: P-values obtained by doing Anderson-Darling normality tests on the t-statistics from the variance tests. The average tail length and average cycle length statistics are excluded because the P-values from these t-tests were consistently very low. Test statistics are unavailable for maximum cycle and maximum tail variance. Tests were run using Minitab.

5 Conclusions and Future Work

The results produced by these tests lend some statistical backing to the statement that the average number of components, cyclic nodes, and max cycle length in functional graphs generated using (1) are similar to the expected statistics for random binary functional graphs. The results also let one say with some degree of confidence that the average cycle length and the average tail length of graphs made with (1) are similar to those random graphs. The most interesting result, however, is the extremely low variance in these two statistics when compared to the expected variance. Unfortunately there is not an immediately evident reason for this, so future work could include an investigation as to why the variance is so much lower, but still somewhat consistent for these statistics. Regression tests could be performed to try and match some sort of function which relates the size of a graph and its observed variance. An initial look at the data, however, makes it appear that such a function would have to be very complicated in order to account for the large jumps that can be seen in the observed results. It is still possible that a relatively simple asymptotic formula exists though.

Intuition seems to indicate that while the choice of the average tail/cycle length for some graph generated by the code may be like what a random binary functional graph is like, it may be that (1) imposes some structure that keeps the tail/cycle length from varying too much between the graphs for
a given prime. Additional research could also be done to attempt formulate an attack on some system that uses (1) based on the knowledge that the average tail or cycle variance will be lower than random graphs. Perhaps, for instance, if one could figure out the prime a pseudorandom bit generator based on (1) is using, he/she could gain more knowledge about how long the generator will take to enter a cycle and repeat than should be available.

While less dramatic, the lower than average maximum tail length is also curious. While a recursive formula was not found for this statistic, the asymptotic analysis does not change significantly when additional terms are added, so it appears that the difference is not due to the asymptotic estimation. Besides being an asymptotic formula, the computation also uses an integral estimate which could also be off slightly. Additional research could be done here to analyze the lower value, but the reason is probably more subtle and does not appear to be as significant.

Theoretical values for the variance in the maximum tail length and maximum cycle length also remain unsolved. These statistics could give further insight into how closely graphs generated with (1) exhibit characteristics of random graphs. Results in this area might also serve to motivate or direct research into why the variance is so low for average cycle/tail lengths.

A number of open problems from [2] also remain unsolved. It is possible to extend this work to deal with any \( m \)-ary graph, but the methods would need to be altered somewhat in order to account for the problems the new variable \( m \) would introduce. It was also pointed out that in practice (for the RSA encryption standard, for instance), sometimes composites of large primes are used instead of primes. Additional research could be done to investigate what graphs generated using this composite modulus look like.

**Acknowledgments** The author would like to thank Joshua Holden for all his great advice and help with the project, and Mark Inlow for all the help with the statistical tests and interpreting the results.
## A Number of Graphs

<table>
<thead>
<tr>
<th>Prime</th>
<th>Number of Graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>99923</td>
<td>48852</td>
</tr>
<tr>
<td>99961</td>
<td>10752</td>
</tr>
<tr>
<td>99971</td>
<td>36864</td>
</tr>
<tr>
<td>99989</td>
<td>21420</td>
</tr>
<tr>
<td>99991</td>
<td>24000</td>
</tr>
<tr>
<td>100003</td>
<td>28560</td>
</tr>
<tr>
<td>100019</td>
<td>48804</td>
</tr>
<tr>
<td>100043</td>
<td>50020</td>
</tr>
<tr>
<td>100049</td>
<td>22464</td>
</tr>
<tr>
<td>100057</td>
<td>15120</td>
</tr>
<tr>
<td>100069</td>
<td>16080</td>
</tr>
<tr>
<td>100103</td>
<td>50050</td>
</tr>
<tr>
<td>106261</td>
<td>10560</td>
</tr>
<tr>
<td>110017</td>
<td>18240</td>
</tr>
<tr>
<td>110459</td>
<td>55228</td>
</tr>
<tr>
<td>120041</td>
<td>24000</td>
</tr>
<tr>
<td>120167</td>
<td>60082</td>
</tr>
<tr>
<td>130021</td>
<td>15680</td>
</tr>
<tr>
<td>130127</td>
<td>65062</td>
</tr>
<tr>
<td>140053</td>
<td>21200</td>
</tr>
<tr>
<td>140123</td>
<td>70060</td>
</tr>
<tr>
<td>150001</td>
<td>20000</td>
</tr>
<tr>
<td>150083</td>
<td>75040</td>
</tr>
<tr>
<td>160009</td>
<td>25984</td>
</tr>
<tr>
<td>160079</td>
<td>80038</td>
</tr>
<tr>
<td>170099</td>
<td>85048</td>
</tr>
<tr>
<td>170101</td>
<td>19440</td>
</tr>
<tr>
<td>180001</td>
<td>24000</td>
</tr>
<tr>
<td>180023</td>
<td>90010</td>
</tr>
<tr>
<td>190093</td>
<td>25920</td>
</tr>
<tr>
<td>190523</td>
<td>95260</td>
</tr>
<tr>
<td>200041</td>
<td>26656</td>
</tr>
<tr>
<td>200087</td>
<td>100042</td>
</tr>
</tbody>
</table>

Table 5: Number of binary functional graphs generated for each prime.
## B Components

<table>
<thead>
<tr>
<th>Prime</th>
<th>Predicted</th>
<th>Observed</th>
<th>t-value</th>
<th>P-value</th>
<th>Predicted</th>
<th>Observed</th>
<th>t-value</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>99923</td>
<td>6.391</td>
<td>6.375</td>
<td>1.559</td>
<td>0.134</td>
<td>5.158</td>
<td>5.170</td>
<td>-0.344</td>
<td>0.764</td>
</tr>
<tr>
<td>99961</td>
<td>6.391</td>
<td>6.384</td>
<td>0.329</td>
<td>0.764</td>
<td>5.158</td>
<td>5.210</td>
<td>-0.702</td>
<td>0.484</td>
</tr>
<tr>
<td>99971</td>
<td>6.391</td>
<td>6.391</td>
<td>0.059</td>
<td>1.000</td>
<td>5.158</td>
<td>5.219</td>
<td>-1.513</td>
<td>0.134</td>
</tr>
<tr>
<td>99989</td>
<td>6.392</td>
<td>6.388</td>
<td>0.261</td>
<td>0.842</td>
<td>5.158</td>
<td>5.147</td>
<td>0.213</td>
<td>0.842</td>
</tr>
<tr>
<td>99991</td>
<td>6.392</td>
<td>6.411</td>
<td>-1.330</td>
<td>0.194</td>
<td>5.158</td>
<td>5.131</td>
<td>0.567</td>
<td>0.618</td>
</tr>
<tr>
<td>10003</td>
<td>6.392</td>
<td>6.415</td>
<td>-1.746</td>
<td>0.090</td>
<td>5.158</td>
<td>5.208</td>
<td>-1.111</td>
<td>0.272</td>
</tr>
<tr>
<td>10019</td>
<td>6.392</td>
<td>6.393</td>
<td>-0.160</td>
<td>0.920</td>
<td>5.158</td>
<td>5.198</td>
<td>-1.136</td>
<td>0.272</td>
</tr>
<tr>
<td>10043</td>
<td>6.392</td>
<td>6.389</td>
<td>0.266</td>
<td>0.842</td>
<td>5.158</td>
<td>5.117</td>
<td>1.227</td>
<td>0.230</td>
</tr>
<tr>
<td>10049</td>
<td>6.392</td>
<td>6.356</td>
<td>2.342</td>
<td>0.022</td>
<td>5.158</td>
<td>5.144</td>
<td>0.272</td>
<td>0.842</td>
</tr>
<tr>
<td>10057</td>
<td>6.392</td>
<td>6.364</td>
<td>1.505</td>
<td>0.134</td>
<td>5.158</td>
<td>5.098</td>
<td>0.981</td>
<td>0.368</td>
</tr>
<tr>
<td>10069</td>
<td>6.392</td>
<td>6.380</td>
<td>0.661</td>
<td>0.548</td>
<td>5.158</td>
<td>5.139</td>
<td>0.316</td>
<td>0.764</td>
</tr>
<tr>
<td>10103</td>
<td>6.392</td>
<td>6.386</td>
<td>0.637</td>
<td>0.548</td>
<td>5.158</td>
<td>5.182</td>
<td>-0.683</td>
<td>0.548</td>
</tr>
<tr>
<td>106261</td>
<td>6.422</td>
<td>6.370</td>
<td>2.350</td>
<td>0.022</td>
<td>5.188</td>
<td>5.176</td>
<td>0.169</td>
<td>0.920</td>
</tr>
<tr>
<td>110017</td>
<td>6.439</td>
<td>6.478</td>
<td>-2.269</td>
<td>0.028</td>
<td>5.206</td>
<td>5.171</td>
<td>0.611</td>
<td>0.548</td>
</tr>
<tr>
<td>110459</td>
<td>6.441</td>
<td>6.453</td>
<td>-1.201</td>
<td>0.230</td>
<td>5.208</td>
<td>5.184</td>
<td>0.738</td>
<td>0.484</td>
</tr>
<tr>
<td>120041</td>
<td>6.483</td>
<td>6.480</td>
<td>0.232</td>
<td>0.842</td>
<td>5.249</td>
<td>5.266</td>
<td>-0.328</td>
<td>0.764</td>
</tr>
<tr>
<td>120167</td>
<td>6.483</td>
<td>6.480</td>
<td>0.324</td>
<td>0.764</td>
<td>5.250</td>
<td>5.207</td>
<td>1.371</td>
<td>0.194</td>
</tr>
<tr>
<td>130021</td>
<td>6.523</td>
<td>6.508</td>
<td>0.801</td>
<td>0.424</td>
<td>5.289</td>
<td>5.170</td>
<td>1.974</td>
<td>0.058</td>
</tr>
<tr>
<td>130127</td>
<td>6.523</td>
<td>6.528</td>
<td>-0.515</td>
<td>0.618</td>
<td>5.290</td>
<td>5.300</td>
<td>-0.352</td>
<td>0.764</td>
</tr>
<tr>
<td>140053</td>
<td>6.560</td>
<td>6.576</td>
<td>-1.031</td>
<td>0.318</td>
<td>5.326</td>
<td>5.298</td>
<td>0.530</td>
<td>0.618</td>
</tr>
<tr>
<td>140123</td>
<td>6.560</td>
<td>6.545</td>
<td>1.745</td>
<td>0.090</td>
<td>5.327</td>
<td>5.324</td>
<td>0.101</td>
<td>0.920</td>
</tr>
<tr>
<td>150001</td>
<td>6.594</td>
<td>6.546</td>
<td>2.978</td>
<td>0.004</td>
<td>5.361</td>
<td>5.354</td>
<td>0.118</td>
<td>0.920</td>
</tr>
<tr>
<td>150083</td>
<td>6.595</td>
<td>6.609</td>
<td>-1.756</td>
<td>0.090</td>
<td>5.361</td>
<td>5.360</td>
<td>0.023</td>
<td>1.000</td>
</tr>
<tr>
<td>160099</td>
<td>6.627</td>
<td>6.600</td>
<td>1.866</td>
<td>0.072</td>
<td>5.393</td>
<td>5.376</td>
<td>0.335</td>
<td>0.764</td>
</tr>
<tr>
<td>160079</td>
<td>6.627</td>
<td>6.637</td>
<td>-1.242</td>
<td>0.230</td>
<td>5.393</td>
<td>5.380</td>
<td>0.472</td>
<td>0.690</td>
</tr>
<tr>
<td>170099</td>
<td>6.657</td>
<td>6.644</td>
<td>1.609</td>
<td>0.110</td>
<td>5.424</td>
<td>5.407</td>
<td>0.596</td>
<td>0.618</td>
</tr>
<tr>
<td>170101</td>
<td>6.657</td>
<td>6.632</td>
<td>1.521</td>
<td>0.134</td>
<td>5.424</td>
<td>5.381</td>
<td>0.738</td>
<td>0.484</td>
</tr>
<tr>
<td>180001</td>
<td>6.686</td>
<td>6.627</td>
<td>3.908</td>
<td>0.000</td>
<td>5.452</td>
<td>5.338</td>
<td>2.247</td>
<td>0.028</td>
</tr>
<tr>
<td>180023</td>
<td>6.686</td>
<td>6.685</td>
<td>0.988</td>
<td>1.000</td>
<td>5.452</td>
<td>5.486</td>
<td>-1.278</td>
<td>0.230</td>
</tr>
<tr>
<td>190093</td>
<td>6.713</td>
<td>6.699</td>
<td>0.985</td>
<td>0.368</td>
<td>5.479</td>
<td>5.444</td>
<td>0.722</td>
<td>0.484</td>
</tr>
<tr>
<td>190523</td>
<td>6.714</td>
<td>6.723</td>
<td>-1.203</td>
<td>0.230</td>
<td>5.480</td>
<td>5.515</td>
<td>-1.339</td>
<td>0.194</td>
</tr>
<tr>
<td>200041</td>
<td>6.738</td>
<td>6.704</td>
<td>2.423</td>
<td>0.016</td>
<td>5.505</td>
<td>5.462</td>
<td>0.872</td>
<td>0.424</td>
</tr>
<tr>
<td>200087</td>
<td>6.738</td>
<td>6.745</td>
<td>-0.919</td>
<td>0.368</td>
<td>5.505</td>
<td>5.517</td>
<td>-0.477</td>
<td>0.690</td>
</tr>
</tbody>
</table>

Table 6: Observed and theoretical mean number of components and variance in number of components for 33 primes. t-statistics are from applying a 2-tail t-test, and were used to obtain the P-value.
### Cyclic Nodes

<table>
<thead>
<tr>
<th>Prime</th>
<th>Predicted</th>
<th>Observed</th>
<th>t-value</th>
<th>P-value</th>
<th>Predicted</th>
<th>Observed</th>
<th>t-value</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>99923</td>
<td>395.179</td>
<td>395.621</td>
<td>-0.472</td>
<td>0.690</td>
<td>42491.925</td>
<td>42783.286</td>
<td>-1.010</td>
<td>0.318</td>
</tr>
<tr>
<td>99961</td>
<td>395.254</td>
<td>392.525</td>
<td>1.372</td>
<td>0.194</td>
<td>42508.159</td>
<td>42563.770</td>
<td>-0.093</td>
<td>1.000</td>
</tr>
<tr>
<td>99971</td>
<td>395.274</td>
<td>390.225</td>
<td>-0.878</td>
<td>0.424</td>
<td>42512.432</td>
<td>43214.886</td>
<td>-2.030</td>
<td>0.046</td>
</tr>
<tr>
<td>99989</td>
<td>395.310</td>
<td>395.114</td>
<td>0.137</td>
<td>0.920</td>
<td>42520.122</td>
<td>43451.612</td>
<td>-2.129</td>
<td>0.036</td>
</tr>
<tr>
<td>99991</td>
<td>395.314</td>
<td>394.853</td>
<td>0.348</td>
<td>0.764</td>
<td>42520.976</td>
<td>42190.923</td>
<td>0.804</td>
<td>0.424</td>
</tr>
<tr>
<td>100003</td>
<td>395.338</td>
<td>396.658</td>
<td>-1.079</td>
<td>0.318</td>
<td>42526.103</td>
<td>42733.314</td>
<td>-0.551</td>
<td>0.618</td>
</tr>
<tr>
<td>100043</td>
<td>395.417</td>
<td>395.303</td>
<td>0.123</td>
<td>0.920</td>
<td>42543.192</td>
<td>42227.348</td>
<td>1.123</td>
<td>0.272</td>
</tr>
<tr>
<td>100048</td>
<td>395.429</td>
<td>395.312</td>
<td>0.085</td>
<td>1.000</td>
<td>42545.755</td>
<td>42136.548</td>
<td>1.536</td>
<td>0.272</td>
</tr>
<tr>
<td>110017</td>
<td>414.708</td>
<td>414.603</td>
<td>0.066</td>
<td>1.000</td>
<td>46804.777</td>
<td>46880.846</td>
<td>-0.146</td>
<td>0.920</td>
</tr>
<tr>
<td>110459</td>
<td>415.543</td>
<td>414.677</td>
<td>-1.663</td>
<td>0.110</td>
<td>46993.651</td>
<td>47011.789</td>
<td>-0.061</td>
<td>1.000</td>
</tr>
<tr>
<td>120041</td>
<td>433.234</td>
<td>433.334</td>
<td>-0.069</td>
<td>1.000</td>
<td>51088.590</td>
<td>50971.173</td>
<td>0.242</td>
<td>0.842</td>
</tr>
<tr>
<td>120167</td>
<td>433.462</td>
<td>433.045</td>
<td>0.452</td>
<td>0.690</td>
<td>51142.441</td>
<td>51228.171</td>
<td>-0.275</td>
<td>0.842</td>
</tr>
<tr>
<td>130021</td>
<td>450.924</td>
<td>453.199</td>
<td>-1.198</td>
<td>0.272</td>
<td>55354.352</td>
<td>56512.667</td>
<td>-1.780</td>
<td>0.090</td>
</tr>
<tr>
<td>140053</td>
<td>468.035</td>
<td>467.198</td>
<td>0.504</td>
<td>0.618</td>
<td>59643.012</td>
<td>58631.769</td>
<td>1.676</td>
<td>0.110</td>
</tr>
<tr>
<td>150001</td>
<td>484.407</td>
<td>483.525</td>
<td>0.494</td>
<td>0.690</td>
<td>63896.358</td>
<td>63807.597</td>
<td>0.130</td>
<td>0.920</td>
</tr>
<tr>
<td>160009</td>
<td>500.339</td>
<td>500.424</td>
<td>-0.053</td>
<td>1.000</td>
<td>68175.897</td>
<td>67755.971</td>
<td>0.674</td>
<td>0.548</td>
</tr>
<tr>
<td>160079</td>
<td>500.449</td>
<td>500.681</td>
<td>-0.252</td>
<td>0.842</td>
<td>68205.832</td>
<td>67829.395</td>
<td>1.047</td>
<td>0.318</td>
</tr>
<tr>
<td>170099</td>
<td>515.904</td>
<td>512.754</td>
<td>3.425</td>
<td>0.000</td>
<td>72490.997</td>
<td>71981.329</td>
<td>1.373</td>
<td>0.194</td>
</tr>
<tr>
<td>170101</td>
<td>515.907</td>
<td>514.844</td>
<td>0.555</td>
<td>0.618</td>
<td>72491.852</td>
<td>71368.015</td>
<td>1.459</td>
<td>0.162</td>
</tr>
<tr>
<td>180001</td>
<td>530.737</td>
<td>529.276</td>
<td>0.823</td>
<td>0.424</td>
<td>76726.139</td>
<td>75536.984</td>
<td>1.640</td>
<td>0.110</td>
</tr>
<tr>
<td>180023</td>
<td>530.769</td>
<td>531.465</td>
<td>-0.753</td>
<td>0.484</td>
<td>76735.549</td>
<td>76786.860</td>
<td>-0.134</td>
<td>0.920</td>
</tr>
<tr>
<td>190003</td>
<td>545.440</td>
<td>542.939</td>
<td>1.416</td>
<td>0.162</td>
<td>81042.959</td>
<td>80875.321</td>
<td>0.222</td>
<td>0.842</td>
</tr>
<tr>
<td>190523</td>
<td>546.058</td>
<td>547.976</td>
<td>-2.079</td>
<td>0.046</td>
<td>81226.899</td>
<td>81064.068</td>
<td>0.416</td>
<td>0.690</td>
</tr>
<tr>
<td>200041</td>
<td>559.556</td>
<td>557.622</td>
<td>1.083</td>
<td>0.318</td>
<td>85298.562</td>
<td>84995.931</td>
<td>0.388</td>
<td>0.764</td>
</tr>
<tr>
<td>200087</td>
<td>559.620</td>
<td>562.252</td>
<td>-2.841</td>
<td>0.006</td>
<td>85318.241</td>
<td>85825.849</td>
<td>-1.258</td>
<td>0.230</td>
</tr>
</tbody>
</table>

Table 7: Observed and theoretical mean number of cyclic nodes and variance in number of cyclic nodes for 33 primes. t-statistics are from applying a 2-tail t-test, and were used to obtain the P-value.
### Average Cycle Length

<table>
<thead>
<tr>
<th>Prime</th>
<th>Predicted</th>
<th>Observed</th>
<th>t-value</th>
<th>P-value</th>
<th>Predicted</th>
<th>Observed</th>
<th>t-value</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>99923</td>
<td>198.090</td>
<td>198.892</td>
<td>-1.236</td>
<td>0.230</td>
<td>27177.770</td>
<td>20631.551</td>
<td>32.900</td>
<td>0.000</td>
</tr>
<tr>
<td>99961</td>
<td>198.127</td>
<td>195.151</td>
<td>2.190</td>
<td>0.036</td>
<td>27188.143</td>
<td>19855.515</td>
<td>18.353</td>
<td>0.000</td>
</tr>
<tr>
<td>99971</td>
<td>198.137</td>
<td>197.991</td>
<td>0.195</td>
<td>0.920</td>
<td>27190.873</td>
<td>20575.488</td>
<td>28.372</td>
<td>0.000</td>
</tr>
<tr>
<td>99991</td>
<td>198.155</td>
<td>198.157</td>
<td>-0.002</td>
<td>1.000</td>
<td>27195.786</td>
<td>20983.340</td>
<td>20.199</td>
<td>0.000</td>
</tr>
<tr>
<td>10003</td>
<td>198.169</td>
<td>198.300</td>
<td>0.154</td>
<td>0.920</td>
<td>27203.976</td>
<td>20983.340</td>
<td>25.561</td>
<td>0.000</td>
</tr>
<tr>
<td>10007</td>
<td>198.185</td>
<td>197.841</td>
<td>0.529</td>
<td>0.618</td>
<td>27203.976</td>
<td>20983.340</td>
<td>32.799</td>
<td>0.000</td>
</tr>
<tr>
<td>10019</td>
<td>198.208</td>
<td>198.319</td>
<td>0.173</td>
<td>0.920</td>
<td>27210.527</td>
<td>20392.727</td>
<td>35.112</td>
<td>0.000</td>
</tr>
<tr>
<td>10043</td>
<td>198.214</td>
<td>197.505</td>
<td>0.748</td>
<td>0.484</td>
<td>27212.165</td>
<td>20198.515</td>
<td>18.353</td>
<td>0.000</td>
</tr>
<tr>
<td>10057</td>
<td>198.222</td>
<td>198.215</td>
<td>-0.002</td>
<td>1.000</td>
<td>27214.349</td>
<td>20680.648</td>
<td>28.372</td>
<td>0.000</td>
</tr>
<tr>
<td>10069</td>
<td>198.234</td>
<td>197.841</td>
<td>0.529</td>
<td>0.618</td>
<td>27203.976</td>
<td>20983.340</td>
<td>32.799</td>
<td>0.000</td>
</tr>
<tr>
<td>100103</td>
<td>198.268</td>
<td>197.704</td>
<td>0.883</td>
<td>0.424</td>
<td>27226.906</td>
<td>20389.102</td>
<td>35.112</td>
<td>0.000</td>
</tr>
<tr>
<td>10061</td>
<td>204.275</td>
<td>206.612</td>
<td>-1.619</td>
<td>0.110</td>
<td>29007.991</td>
<td>22003.465</td>
<td>24.740</td>
<td>0.000</td>
</tr>
<tr>
<td>10107</td>
<td>207.854</td>
<td>206.185</td>
<td>0.154</td>
<td>0.920</td>
<td>29933.434</td>
<td>22186.572</td>
<td>24.445</td>
<td>0.000</td>
</tr>
<tr>
<td>101459</td>
<td>208.271</td>
<td>209.063</td>
<td>-1.228</td>
<td>0.230</td>
<td>30054.110</td>
<td>22921.145</td>
<td>33.984</td>
<td>0.000</td>
</tr>
<tr>
<td>120041</td>
<td>217.117</td>
<td>216.709</td>
<td>0.399</td>
<td>0.764</td>
<td>32670.422</td>
<td>25058.237</td>
<td>21.701</td>
<td>0.000</td>
</tr>
<tr>
<td>120167</td>
<td>217.231</td>
<td>216.963</td>
<td>0.418</td>
<td>0.690</td>
<td>32704.828</td>
<td>24617.970</td>
<td>22.445</td>
<td>0.000</td>
</tr>
<tr>
<td>130021</td>
<td>225.962</td>
<td>227.173</td>
<td>-0.925</td>
<td>0.368</td>
<td>35395.773</td>
<td>26875.914</td>
<td>19.489</td>
<td>0.000</td>
</tr>
<tr>
<td>130127</td>
<td>226.054</td>
<td>227.404</td>
<td>-2.090</td>
<td>0.046</td>
<td>35424.722</td>
<td>27112.730</td>
<td>35.060</td>
<td>0.000</td>
</tr>
<tr>
<td>140053</td>
<td>234.518</td>
<td>235.155</td>
<td>-0.546</td>
<td>0.618</td>
<td>38135.661</td>
<td>28874.204</td>
<td>21.605</td>
<td>0.000</td>
</tr>
<tr>
<td>140123</td>
<td>234.576</td>
<td>234.477</td>
<td>0.154</td>
<td>0.920</td>
<td>38154.780</td>
<td>28860.896</td>
<td>40.497</td>
<td>0.000</td>
</tr>
<tr>
<td>150001</td>
<td>242.704</td>
<td>244.271</td>
<td>-1.255</td>
<td>0.230</td>
<td>40852.904</td>
<td>31212.642</td>
<td>20.560</td>
<td>0.000</td>
</tr>
<tr>
<td>150083</td>
<td>242.770</td>
<td>244.601</td>
<td>-2.846</td>
<td>0.006</td>
<td>40875.303</td>
<td>31056.500</td>
<td>42.924</td>
<td>0.000</td>
</tr>
<tr>
<td>160099</td>
<td>250.669</td>
<td>253.280</td>
<td>-2.292</td>
<td>0.028</td>
<td>43586.806</td>
<td>33696.107</td>
<td>22.938</td>
<td>0.000</td>
</tr>
<tr>
<td>160079</td>
<td>250.724</td>
<td>250.806</td>
<td>-0.130</td>
<td>1.000</td>
<td>43605.929</td>
<td>32881.529</td>
<td>42.924</td>
<td>0.000</td>
</tr>
<tr>
<td>170099</td>
<td>258.452</td>
<td>256.815</td>
<td>0.534</td>
<td>0.618</td>
<td>46343.356</td>
<td>34436.466</td>
<td>47.313</td>
<td>0.000</td>
</tr>
<tr>
<td>170101</td>
<td>258.454</td>
<td>258.444</td>
<td>0.007</td>
<td>1.000</td>
<td>46343.903</td>
<td>35361.541</td>
<td>22.938</td>
<td>0.000</td>
</tr>
<tr>
<td>180001</td>
<td>265.868</td>
<td>264.323</td>
<td>1.252</td>
<td>0.230</td>
<td>49048.767</td>
<td>36552.212</td>
<td>24.427</td>
<td>0.000</td>
</tr>
<tr>
<td>180023</td>
<td>265.885</td>
<td>266.585</td>
<td>-0.109</td>
<td>0.318</td>
<td>49054.778</td>
<td>37210.351</td>
<td>44.854</td>
<td>0.000</td>
</tr>
<tr>
<td>190093</td>
<td>273.220</td>
<td>273.708</td>
<td>-0.395</td>
<td>0.764</td>
<td>51806.297</td>
<td>39567.572</td>
<td>22.938</td>
<td>0.000</td>
</tr>
<tr>
<td>190523</td>
<td>273.529</td>
<td>274.813</td>
<td>-1.995</td>
<td>0.058</td>
<td>51923.794</td>
<td>39467.061</td>
<td>45.135</td>
<td>0.000</td>
</tr>
<tr>
<td>200041</td>
<td>280.278</td>
<td>279.706</td>
<td>0.460</td>
<td>0.690</td>
<td>54524.668</td>
<td>41233.270</td>
<td>24.104</td>
<td>0.000</td>
</tr>
<tr>
<td>200087</td>
<td>280.310</td>
<td>281.659</td>
<td>-2.097</td>
<td>0.046</td>
<td>54537.238</td>
<td>41358.233</td>
<td>47.640</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Table 8: Observed and theoretical mean average cycle length and variance in average cycle length for 33 primes. t-statistics are from applying a 2-tail t-test, and were used to obtain the P-value.
### Average Tail Length

<table>
<thead>
<tr>
<th>Prime</th>
<th>Predicted</th>
<th>Observed</th>
<th>t-value</th>
<th>P-value</th>
<th>Predicted</th>
<th>Observed</th>
<th>t-value</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>99923</td>
<td>197.094</td>
<td>197.552</td>
<td>-1.177</td>
<td>0.272</td>
<td>27178.199</td>
<td>7400.315</td>
<td>289.163</td>
<td>0.000</td>
</tr>
<tr>
<td>99961</td>
<td>197.131</td>
<td>196.782</td>
<td>0.424</td>
<td>0.690</td>
<td>27188.572</td>
<td>7279.408</td>
<td>135.068</td>
<td>0.000</td>
</tr>
<tr>
<td>99971</td>
<td>197.141</td>
<td>198.754</td>
<td>-3.563</td>
<td>0.000</td>
<td>27191.302</td>
<td>7554.003</td>
<td>242.988</td>
<td>0.000</td>
</tr>
<tr>
<td>99989</td>
<td>197.159</td>
<td>197.091</td>
<td>0.116</td>
<td>0.920</td>
<td>27196.215</td>
<td>7337.799</td>
<td>194.463</td>
<td>0.000</td>
</tr>
<tr>
<td>99991</td>
<td>197.161</td>
<td>196.741</td>
<td>0.763</td>
<td>0.484</td>
<td>27196.761</td>
<td>7263.687</td>
<td>216.992</td>
<td>0.000</td>
</tr>
<tr>
<td>100003</td>
<td>197.173</td>
<td>197.926</td>
<td>-1.492</td>
<td>0.162</td>
<td>27200.037</td>
<td>7269.436</td>
<td>228.496</td>
<td>0.000</td>
</tr>
<tr>
<td>100019</td>
<td>197.189</td>
<td>197.071</td>
<td>0.303</td>
<td>0.764</td>
<td>27204.405</td>
<td>7401.001</td>
<td>289.771</td>
<td>0.000</td>
</tr>
<tr>
<td>100043</td>
<td>197.212</td>
<td>197.178</td>
<td>0.088</td>
<td>1.000</td>
<td>27210.956</td>
<td>7362.882</td>
<td>166.087</td>
<td>0.000</td>
</tr>
<tr>
<td>100049</td>
<td>197.218</td>
<td>198.290</td>
<td>-1.859</td>
<td>0.072</td>
<td>27212.594</td>
<td>7464.627</td>
<td>194.361</td>
<td>0.000</td>
</tr>
<tr>
<td>100057</td>
<td>197.226</td>
<td>196.768</td>
<td>0.657</td>
<td>0.548</td>
<td>27214.778</td>
<td>7335.733</td>
<td>160.217</td>
<td>0.000</td>
</tr>
<tr>
<td>100069</td>
<td>197.238</td>
<td>200.522</td>
<td>-4.739</td>
<td>0.000</td>
<td>27218.054</td>
<td>7722.150</td>
<td>162.246</td>
<td>0.000</td>
</tr>
<tr>
<td>110017</td>
<td>206.858</td>
<td>205.797</td>
<td>1.602</td>
<td>0.110</td>
<td>29933.863</td>
<td>8004.221</td>
<td>177.592</td>
<td>0.000</td>
</tr>
<tr>
<td>110049</td>
<td>207.275</td>
<td>207.261</td>
<td>0.036</td>
<td>1.000</td>
<td>32705.257</td>
<td>8165.605</td>
<td>310.309</td>
<td>0.000</td>
</tr>
<tr>
<td>120041</td>
<td>216.121</td>
<td>216.655</td>
<td>-0.877</td>
<td>0.242</td>
<td>32670.851</td>
<td>8910.985</td>
<td>203.307</td>
<td>0.000</td>
</tr>
<tr>
<td>120067</td>
<td>216.235</td>
<td>215.620</td>
<td>1.602</td>
<td>0.110</td>
<td>32705.257</td>
<td>8845.946</td>
<td>325.315</td>
<td>0.000</td>
</tr>
<tr>
<td>130021</td>
<td>224.966</td>
<td>224.368</td>
<td>0.759</td>
<td>0.484</td>
<td>35396.203</td>
<td>9728.814</td>
<td>160.217</td>
<td>0.000</td>
</tr>
<tr>
<td>130053</td>
<td>233.521</td>
<td>234.855</td>
<td>-1.907</td>
<td>0.058</td>
<td>38136.090</td>
<td>10380.536</td>
<td>193.514</td>
<td>0.000</td>
</tr>
<tr>
<td>140053</td>
<td>233.580</td>
<td>233.598</td>
<td>-0.047</td>
<td>1.000</td>
<td>38155.209</td>
<td>10318.550</td>
<td>351.600</td>
<td>0.000</td>
</tr>
<tr>
<td>150001</td>
<td>241.707</td>
<td>242.031</td>
<td>-0.433</td>
<td>0.690</td>
<td>40853.334</td>
<td>11220.811</td>
<td>185.284</td>
<td>0.000</td>
</tr>
<tr>
<td>150083</td>
<td>241.773</td>
<td>241.350</td>
<td>1.107</td>
<td>0.272</td>
<td>40875.733</td>
<td>10941.976</td>
<td>193.514</td>
<td>0.000</td>
</tr>
<tr>
<td>160009</td>
<td>249.673</td>
<td>248.871</td>
<td>1.206</td>
<td>0.230</td>
<td>43587.235</td>
<td>11474.843</td>
<td>221.910</td>
<td>0.000</td>
</tr>
<tr>
<td>160079</td>
<td>249.727</td>
<td>249.619</td>
<td>0.283</td>
<td>0.842</td>
<td>43606.358</td>
<td>11726.168</td>
<td>374.063</td>
<td>0.000</td>
</tr>
<tr>
<td>170099</td>
<td>257.455</td>
<td>257.153</td>
<td>0.791</td>
<td>0.484</td>
<td>46343.786</td>
<td>12382.481</td>
<td>202.746</td>
<td>0.000</td>
</tr>
<tr>
<td>170101</td>
<td>257.457</td>
<td>257.594</td>
<td>-0.172</td>
<td>0.920</td>
<td>46344.332</td>
<td>12394.279</td>
<td>187.779</td>
<td>0.000</td>
</tr>
<tr>
<td>180001</td>
<td>264.871</td>
<td>265.199</td>
<td>-0.440</td>
<td>0.690</td>
<td>49049.196</td>
<td>13336.878</td>
<td>202.746</td>
<td>0.000</td>
</tr>
<tr>
<td>180023</td>
<td>264.888</td>
<td>265.843</td>
<td>-2.468</td>
<td>0.016</td>
<td>49055.207</td>
<td>13477.276</td>
<td>385.937</td>
<td>0.000</td>
</tr>
<tr>
<td>190093</td>
<td>272.232</td>
<td>273.566</td>
<td>-1.822</td>
<td>0.072</td>
<td>51806.726</td>
<td>14080.980</td>
<td>213.220</td>
<td>0.000</td>
</tr>
<tr>
<td>190523</td>
<td>272.532</td>
<td>273.337</td>
<td>-2.094</td>
<td>0.046</td>
<td>51924.223</td>
<td>14072.768</td>
<td>412.570</td>
<td>0.000</td>
</tr>
<tr>
<td>200041</td>
<td>279.281</td>
<td>280.272</td>
<td>-1.338</td>
<td>0.194</td>
<td>54525.097</td>
<td>14637.299</td>
<td>224.727</td>
<td>0.000</td>
</tr>
<tr>
<td>200087</td>
<td>279.313</td>
<td>278.974</td>
<td>0.883</td>
<td>0.424</td>
<td>54537.668</td>
<td>14731.689</td>
<td>418.880</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Table 9: Observed and theoretical mean average tail length and variance in average tail length for 33 primes. t-statistics are from applying a 2-tail t-test, and were used to obtain the P-value.
### Maximum Cycle

<table>
<thead>
<tr>
<th>Prime</th>
<th>Predicted</th>
<th>Mean</th>
<th>Variance</th>
<th>t-value</th>
<th>P-value</th>
<th>Observed</th>
<th>t-value</th>
<th>P-value</th>
<th>Observed</th>
</tr>
</thead>
<tbody>
<tr>
<td>99923</td>
<td>247.347</td>
<td>247.597</td>
<td>-0.358</td>
<td>0.764</td>
<td>24099.258</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99961</td>
<td>247.394</td>
<td>244.195</td>
<td>2.176</td>
<td>0.036</td>
<td>23231.014</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99971</td>
<td>247.406</td>
<td>247.104</td>
<td>0.375</td>
<td>0.764</td>
<td>24003.558</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99999</td>
<td>247.428</td>
<td>246.834</td>
<td>0.557</td>
<td>0.618</td>
<td>24354.597</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99991</td>
<td>247.431</td>
<td>246.591</td>
<td>0.841</td>
<td>0.424</td>
<td>23880.666</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100003</td>
<td>247.446</td>
<td>247.461</td>
<td>-0.017</td>
<td>1.000</td>
<td>23981.212</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100019</td>
<td>247.465</td>
<td>246.537</td>
<td>1.327</td>
<td>0.194</td>
<td>23881.212</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100043</td>
<td>247.495</td>
<td>247.261</td>
<td>0.339</td>
<td>0.764</td>
<td>23806.218</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100049</td>
<td>247.502</td>
<td>246.768</td>
<td>0.716</td>
<td>0.484</td>
<td>23652.104</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100057</td>
<td>247.512</td>
<td>247.302</td>
<td>0.167</td>
<td>0.920</td>
<td>23885.249</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100069</td>
<td>247.527</td>
<td>246.502</td>
<td>0.839</td>
<td>0.424</td>
<td>24303.513</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100103</td>
<td>247.569</td>
<td>246.561</td>
<td>1.464</td>
<td>0.162</td>
<td>23746.781</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>106261</td>
<td>255.070</td>
<td>256.986</td>
<td>-1.229</td>
<td>0.230</td>
<td>25629.420</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110017</td>
<td>259.539</td>
<td>257.916</td>
<td>1.362</td>
<td>0.194</td>
<td>25908.517</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110459</td>
<td>260.060</td>
<td>260.661</td>
<td>-0.867</td>
<td>0.424</td>
<td>26503.298</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>120041</td>
<td>271.105</td>
<td>270.519</td>
<td>0.534</td>
<td>0.618</td>
<td>28918.058</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>120167</td>
<td>271.248</td>
<td>270.751</td>
<td>0.718</td>
<td>0.484</td>
<td>28760.455</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>130021</td>
<td>282.150</td>
<td>283.448</td>
<td>-0.917</td>
<td>0.368</td>
<td>31463.383</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>130127</td>
<td>282.265</td>
<td>283.567</td>
<td>-1.873</td>
<td>0.072</td>
<td>31441.827</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>140053</td>
<td>292.833</td>
<td>292.879</td>
<td>-0.037</td>
<td>1.000</td>
<td>33562.989</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>140123</td>
<td>292.906</td>
<td>292.427</td>
<td>0.691</td>
<td>0.548</td>
<td>33621.248</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150001</td>
<td>303.054</td>
<td>303.790</td>
<td>-0.547</td>
<td>0.618</td>
<td>36165.692</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150083</td>
<td>303.137</td>
<td>304.979</td>
<td>-2.659</td>
<td>0.010</td>
<td>35994.045</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160009</td>
<td>313.001</td>
<td>315.267</td>
<td>-1.855</td>
<td>0.072</td>
<td>38751.775</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160079</td>
<td>313.069</td>
<td>312.615</td>
<td>0.659</td>
<td>0.548</td>
<td>38105.498</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>170099</td>
<td>322.719</td>
<td>320.966</td>
<td>3.525</td>
<td>0.000</td>
<td>40178.418</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>170101</td>
<td>322.721</td>
<td>321.416</td>
<td>0.901</td>
<td>0.368</td>
<td>40801.664</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180001</td>
<td>331.979</td>
<td>330.325</td>
<td>1.242</td>
<td>0.230</td>
<td>42612.976</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180023</td>
<td>332.000</td>
<td>332.393</td>
<td>-0.567</td>
<td>0.618</td>
<td>43236.317</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>190093</td>
<td>341.159</td>
<td>340.368</td>
<td>0.595</td>
<td>0.618</td>
<td>45900.280</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>190523</td>
<td>341.544</td>
<td>342.689</td>
<td>-1.650</td>
<td>0.110</td>
<td>45840.435</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200041</td>
<td>349.972</td>
<td>348.661</td>
<td>0.978</td>
<td>0.368</td>
<td>47826.847</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200087</td>
<td>350.012</td>
<td>351.356</td>
<td>-1.939</td>
<td>0.058</td>
<td>48068.838</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 10: Observed and theoretical mean maximum cycle and observed variance in maximum cycle length for 33 primes. t-statistics are from applying a 2-tail t-test, and were used to obtain the P-value. No theoretical values for maximum cycle variance exist yet.
### Maximum Tail

<table>
<thead>
<tr>
<th>Prime</th>
<th>Predicted</th>
<th>Observed</th>
<th>t-value</th>
<th>P-value</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>99923</td>
<td>547.606</td>
<td>543.281</td>
<td>5.835</td>
<td>0.000</td>
<td>26833.550</td>
</tr>
<tr>
<td>99961</td>
<td>547.710</td>
<td>541.005</td>
<td>4.253</td>
<td>0.000</td>
<td>26730.420</td>
</tr>
<tr>
<td>99971</td>
<td>547.738</td>
<td>544.967</td>
<td>3.219</td>
<td>0.002</td>
<td>26665.133</td>
</tr>
<tr>
<td>99989</td>
<td>547.787</td>
<td>542.476</td>
<td>4.746</td>
<td>0.000</td>
<td>26833.394</td>
</tr>
<tr>
<td>99991</td>
<td>547.793</td>
<td>541.265</td>
<td>6.173</td>
<td>0.000</td>
<td>26833.102</td>
</tr>
<tr>
<td>100003</td>
<td>547.826</td>
<td>543.877</td>
<td>4.087</td>
<td>0.000</td>
<td>26665.133</td>
</tr>
<tr>
<td>100019</td>
<td>547.870</td>
<td>542.008</td>
<td>7.905</td>
<td>0.000</td>
<td>26709.198</td>
</tr>
<tr>
<td>100043</td>
<td>547.935</td>
<td>541.827</td>
<td>8.359</td>
<td>0.000</td>
<td>26709.198</td>
</tr>
<tr>
<td>100049</td>
<td>547.952</td>
<td>544.386</td>
<td>3.226</td>
<td>0.002</td>
<td>27440.183</td>
</tr>
<tr>
<td>100057</td>
<td>547.974</td>
<td>541.701</td>
<td>4.707</td>
<td>0.000</td>
<td>26848.354</td>
</tr>
<tr>
<td>100069</td>
<td>548.007</td>
<td>549.379</td>
<td>-1.049</td>
<td>0.318</td>
<td>27531.440</td>
</tr>
<tr>
<td>100103</td>
<td>548.100</td>
<td>540.967</td>
<td>9.702</td>
<td>0.000</td>
<td>27059.016</td>
</tr>
<tr>
<td>100261</td>
<td>564.756</td>
<td>554.905</td>
<td>6.093</td>
<td>0.000</td>
<td>28531.293</td>
</tr>
<tr>
<td>110017</td>
<td>574.679</td>
<td>565.605</td>
<td>7.192</td>
<td>0.000</td>
<td>29039.726</td>
</tr>
<tr>
<td>110459</td>
<td>575.836</td>
<td>569.769</td>
<td>8.243</td>
<td>0.000</td>
<td>29920.698</td>
</tr>
<tr>
<td>120041</td>
<td>600.361</td>
<td>595.029</td>
<td>4.580</td>
<td>0.000</td>
<td>32531.293</td>
</tr>
<tr>
<td>120167</td>
<td>600.677</td>
<td>592.276</td>
<td>11.482</td>
<td>0.000</td>
<td>32159.836</td>
</tr>
<tr>
<td>130021</td>
<td>624.885</td>
<td>616.087</td>
<td>5.877</td>
<td>0.000</td>
<td>35141.251</td>
</tr>
<tr>
<td>130127</td>
<td>625.141</td>
<td>621.785</td>
<td>4.553</td>
<td>0.000</td>
<td>35347.570</td>
</tr>
<tr>
<td>140053</td>
<td>648.606</td>
<td>645.702</td>
<td>2.169</td>
<td>0.036</td>
<td>38004.996</td>
</tr>
<tr>
<td>140123</td>
<td>648.768</td>
<td>642.440</td>
<td>8.629</td>
<td>0.000</td>
<td>37674.203</td>
</tr>
<tr>
<td>150001</td>
<td>671.303</td>
<td>665.275</td>
<td>4.243</td>
<td>0.000</td>
<td>40375.416</td>
</tr>
<tr>
<td>150083</td>
<td>671.486</td>
<td>665.402</td>
<td>8.280</td>
<td>0.000</td>
<td>40529.603</td>
</tr>
<tr>
<td>160009</td>
<td>693.389</td>
<td>685.142</td>
<td>6.479</td>
<td>0.000</td>
<td>42098.894</td>
</tr>
<tr>
<td>160079</td>
<td>693.541</td>
<td>686.769</td>
<td>9.240</td>
<td>0.000</td>
<td>42989.106</td>
</tr>
<tr>
<td>170099</td>
<td>714.967</td>
<td>707.318</td>
<td>10.494</td>
<td>0.000</td>
<td>45190.679</td>
</tr>
<tr>
<td>170101</td>
<td>714.971</td>
<td>709.991</td>
<td>3.269</td>
<td>0.002</td>
<td>45110.111</td>
</tr>
<tr>
<td>180001</td>
<td>735.529</td>
<td>730.899</td>
<td>3.243</td>
<td>0.002</td>
<td>48941.776</td>
</tr>
<tr>
<td>180023</td>
<td>735.574</td>
<td>731.320</td>
<td>5.766</td>
<td>0.000</td>
<td>48996.888</td>
</tr>
<tr>
<td>190093</td>
<td>755.912</td>
<td>752.173</td>
<td>2.656</td>
<td>0.010</td>
<td>51345.311</td>
</tr>
<tr>
<td>190523</td>
<td>756.768</td>
<td>752.829</td>
<td>5.365</td>
<td>0.000</td>
<td>51364.029</td>
</tr>
<tr>
<td>200041</td>
<td>775.481</td>
<td>770.913</td>
<td>3.207</td>
<td>0.002</td>
<td>54093.098</td>
</tr>
<tr>
<td>200087</td>
<td>775.570</td>
<td>769.207</td>
<td>8.657</td>
<td>0.000</td>
<td>54039.057</td>
</tr>
</tbody>
</table>

Table 11: Observed and theoretical mean maximum tail and observed variance in maximum tail length for 33 primes. t-statistics are from applying a 2-tail t-test, and were used to obtain the P-value. No theoretical values for maximum tail variance exist yet.
H Code

H.1 bnprime.h

/* Auto generated by bn_prime.pl */
/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
 * All rights reserved.
 *
 * This package is an SSL implementation written
 * by Eric Young (eay@cryptsoft.com).
 * The implementation was written so as to conform with Netscapes SSL.
 *
 * This library is free for commercial and non-commercial use as long as
 * the following conditions are aheared to. The following conditions
 * apply to all code found in this distribution, be it the RC4, RSA,
 * lhash, DES, etc., code; not just the SSL code. The SSL documentation
 * included with this distribution is covered by the same copyright terms
 * except that the holder is Tim Hudson (tjh@cryptsoft.com).
 *
 * Copyright remains Eric Young's, and as such any Copyright notices in
 * the code are not to be removed.
 * If this package is used in a product, Eric Young should be given attribution
 * as the author of the parts of the library used.
 * This can be in the form of a textual message at program startup or
 * in documentation (online or textual) provided with the package.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *    "This product includes cryptographic software written by
 *    Eric Young (eay@cryptsoft.com)"
 *    The word 'cryptographic' can be left out if the rouines from the library
 *    being used are not cryptographic related :-).
 * 4. If you include any Windows specific code (or a derivative thereof) from
 *    the apps directory (application code) you must include an acknowledgement:
 *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
 *
 * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ‘‘AS IS’’ AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
*
* The licence and distribution terms for any publically available version or
* derivative of this code cannot be changed. i.e. this code cannot simply be
* copied and put under another distribution licence
* [including the GNU Public Licence.]
*/,

```c
#ifndef EIGHT_BIT
#define NUMPRIMES 2048
#else
#define NUMPRIMES 54
#endif

static const unsigned int primes[NUMPRIMES] =
{
  2, 3, 5, 7, 11, 13, 17, 19,
  23, 29, 31, 37, 41, 43, 47, 53,
  59, 61, 67, 71, 73, 79, 83, 89,
  97, 101, 103, 107, 109, 113, 127, 131,
  137, 139, 149, 151, 157, 163, 167, 173,
  179, 181, 191, 193, 197, 211, 223,
  227, 229, 233, 239, 241, 251,
#else
  257, 263,
  269, 271, 277, 281, 283, 293, 307, 311,
  313, 317, 331, 337, 347, 349, 353, 359,
  367, 373, 379, 383, 389, 397, 401, 409,
  419, 421, 431, 433, 439, 443, 449, 457,
  461, 463, 467, 479, 487, 491, 499, 503,
  509, 521, 523, 541, 547, 557, 563, 569,
  571, 577, 587, 593, 599, 601, 607, 613,
  617, 619, 631, 641, 643, 647, 653, 659,
  661, 673, 677, 683, 691, 701, 709, 719,
  727, 733, 739, 743, 751, 757, 761, 769,
  773, 787, 797, 809, 811, 821, 823, 827,
  829, 839, 853, 857, 859, 863, 877, 881,
  883, 887, 907, 911, 919, 929, 937, 941,
  947, 953, 967, 971, 977, 983, 991, 997
#endif
```

20
H.2 ca3.h

#define CA3

#include <stdio.h>
#include<math.h>
#include "bn_prime.h"
#include "mpi.h"

#define STATUS "status.txt"
#define n 10069
#define trials 1676.0
#define M_ARY 2

/*
  7 -- 2
  11 -- 4
  1019 -- 508
  2579 -- 1288
  10069 -- 1676

  99923 -- 48852
  99961 -- 10752
  99971 -- 36864
  99989 -- 21420
  99991 -- 24000
  100003 -- 28560
  100019 -- 48840
  100049 -- 22464
  100069 -- 16080
  100103 -- 50050

  100057 -- 15120
  100043 -- 50020
  106261 -- 10560
*/
#define bool char
#define false 0
#define true 1

void run();
void run2();
int m_expn(int b, int r, int num);
int m_exp(int b, int r);
long long ml_exp(long long, int, long long);
void computeResults( const int*, const int*, int*, int* );
void setArrays(int*, bool*, int*, bool*);
bool isPrimRoot(int);
bool isRelPrime(int);
void writeTotalResults(int*, int*, int*, int*, int*, int*, int*);

int gcd(int, int);
bool isPrime(int);
bool MillerRabin(int, int, int, int);

#endif

H.3 Lindle2.c
#include "ca3.h"

int main(int argc, char* argv[]) {

    MPI_Init(&argc, &argv);

    run();

    MPI_Finalize();

    return 0;
}
H.4 Lindle2ca.c

#include "ca3.h"

int mynode, totalnodes;

int m_expn(int b, int r, int num) {
    return (int) ml_exp((long long) b, r, (long long) num);
}

int m_exp(int b, int r) {
    return m_expn(b, r, n);
}

/*
long long ml_exp(long long b, int r, long long num) {
if (r == 0) return 1;
if(r % 2 == 0) {
    long long result = ml_exp(b,r/2,num);
    return result * result % num;
}
long long result = ml_exp(b,r/2,num);
return (b * result % num) * result % num;
}
/*
*/
/*taken from wikipedia page*/
long long ml_exp(long long b, int e, long long m) {

    long long result = 1;
    while (e > 0) {
        /* multiply in this bits' contribution while using modulus to keep result small*/
        if ((e & 1) == 1) result = (result * b) % m;
        e >>= 1;
        b = (b * b) % m;
    }
    return result;
}

void computeResults(const int* distToCycle, const int* cycleSize,
int* allToCycleSum, int* allCycleLengthSum) {
int sumDistToCycle = 0;
int sumCycleSize = 0;
int i = 0;
for(i = 0; i < n; i++) {
    sumDistToCycle += distToCycle[i];
    sumCycleSize += cycleSize[i];
}

*allToCycleSum = sumDistToCycle;
*allCycleLengthSum = sumCycleSize;
}

bool isPrimRoot(int base) {
    if(!isPrime(n)) return false;

    int n_1 = n-1;

    if (((unsigned)n_1 > (primes[NUMPRIMES-1]*primes[NUMPRIMES-1])))
        printf("Error in Primitive Root Testing, n could have prime factor too large for testing\n");

    int n1 = n_1;
    int index = 0;
    int p;

    while(n1 > 1 && index < NUMPRIMES) {
        /*find the primes that divide phi(n)*/
        if((n1 % primes[index]) == 0) {
            p = primes[index];
            /* divide out that prime all the way so it isn't tested again */
            while((n1 % primes[index]) == 0) n1/=primes[index];
            /*if base^phi(n)/p is 1, not a prim root */
            if(m_exp(base,n_1/p) == 1) return false;
            /*if(isPrime(n1)) return m_exp(base,n_1/n1) == 1; */
            if(n1 == 50021) return (m_exp(base,n_1/50021) != 1);
        }
        index++;
    }
    return true;
}

bool isPrime(int num) {
    int i = 0;
    for(i = 0; i < 50;i++) {
        if(primes[i] > (unsigned)num) return true;
        if((num % primes[i] == 0) && (num!=primes[i])) return false;
    }
}
int k = 0;
int q = num-1;
while(q % 2 == 0) {
    k++;
    q >>= 1;
}
srand(time(0));
int a;
for(i = 0; i < 10; i++) {
    a = (rand() % (num-2)) + 1;
    if(!MillerRabin(num, k, q, a)) return false;
}
return true;
}

bool MillerRabin(int num, int k, int q, int a) {
    int n1 = num-1;
    if(m_expn(a,q, num) == 1) return true;
    int i = 0;
    for(i = 0; i < k; i++)
        if(m_expn(a,(int)pow(2,i)*q, num) == n1) return true;
    return false;
}

bool isRelPrime(int base) {
    return gcd(base, n-1) == 1;
}

int gcd(int a, int b) {
    if(a== 0) return b;
    if(b==0) return a;
    int r = a % b;
    int d = b;
    int c;
    while (r > 0) {
        c = d;
        d = r;
        r = c % d;
    }
    return d;
}

void setArrays(int * cycleSize, bool* visit, int* distToCycle, bool* image){
int i = 0;
for(i = 0; i < n; i++)
{ 
visit[i] = false;
cycleSize[i] = 0;
distToCycle[i] = 0;
image[i] = false;
}
}

void zeroList(int * listArray) {
int i = 0;
for(i = 0; i < n; i++)
listArray[i] = 0;
}

void writeTotalResults( 
int* maxTAll, 
int* maxCAll, 
int* terminalAll, 
int* allComponents, 
int* allCyclicNodes, 
int* allToCycleSum, 
int* allCycleLengthSum) {

char fileStr[20];
sprintf(fileStr, "%d_%d_%d.dat", n, M_ARY, mynode);
FILE * out = fopen(fileStr, "w");
/** # cycles base i 
// sum of cycle size seen from nodes in base i 
// sum of distance to cycle from nodes in base i 
// terminal nodes for base i 
// max cycle for base i 
// max tail for base i 
// cyclic nodes for base i */
int i = 2;
for(i = 2; i < n; i++) { /*0 and 1 not considered*/
fprintf(out, "%d %d %d %d %d %d\n", allComponents[i], allCycleLengthSum[i],
allToCycleSum[i], terminalAll[i], maxCAll[i], maxTAll[i], allCyclicNodes[i]);
} 
fclose(out);

double cComponents = 0;
double cComponentsSquared = 0;
double cCyclicNodes = 0;
double cCyclicNodesSquared = 0;

double cImageNodes = 0;
/*variance = 0*/

double cMaxCycle = 0;
double cMaxCycleSquared = 0;

double cMaxTail = 0;
double cMaxTailSquared = 0;

double cWeightedCycle = 0;
double cWeightedCycleSquared = 0;

double cWeightedTail = 0;
double cWeightedTailSquared = 0;

for (i = 2; i < n; i++)
{
    cComponents += ((double)allComponents[i]) / trials;
    cComponentsSquared += (double)allComponents[i] * (double)allComponents[i] / trials;

    cCyclicNodes += (double)allCyclicNodes[i] / trials;
    cCyclicNodesSquared += (double)allCyclicNodes[i] * (double)allCyclicNodes[i] / trials;

    double cycle = (double)allCycleLengthSum[i] / (double)(n-1);
    cWeightedCycle += (double)(cycle) / trials;
    cWeightedCycleSquared += (double)(cycle*cycle) / (trials);

    double tail = (double)allToCycleSum[i] / (double)(n-1);
    cWeightedTail += tail / trials;
    cWeightedTailSquared += (double)(tail*tail) / (trials);

    if (i != n)
    {
        if (terminalAll[i] > 0)
            cImageNodes += ((double)(n-1) - terminalAll[i]) / trials;
        cMaxCycle += (double)maxCAll[i] / trials;
        cMaxCycleSquared += (double)maxCAll[i]*(double)maxCAll[i] / trials;

        cMaxTail += (double)maxTAll[i] / trials;
        cMaxTailSquared += (double)maxTAll[i]*(double)maxTAll[i] / trials;
    }
}
double cComponentsTot = 0;
double cComponentsSquaredTot = 0;

double cCyclicNodesTot = 0;
double cCyclicNodesSquaredTot = 0;

double cImageNodesTot = 0;
double cMaxCycleTot = 0;
double cMaxCycleSquaredTot = 0;

double cMaxTailTot = 0;
double cMaxTailSquaredTot = 0;

double cWeightedCycleTot = 0;
double cWeightedCycleSquaredTot = 0;

double cWeightedTailTot = 0;
double cWeightedTailSquaredTot = 0;

MPI_Reduce( &cComponents, &cComponentsTot, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
MPI_Reduce( &cComponentsSquared, &cComponentsSquaredTot, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

MPI_Reduce( &cCyclicNodes, &cCyclicNodesTot, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
MPI_Reduce( &cCyclicNodesSquared, &cCyclicNodesSquaredTot, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

MPI_Reduce( &cImageNodes, &cImageNodesTot, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

MPI_Reduce( &cMaxCycle, &cMaxCycleTot, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
MPI_Reduce( &cMaxCycleSquared, &cMaxCycleSquaredTot, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

MPI_Reduce( &cMaxTail, &cMaxTailTot, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
MPI_Reduce( &cMaxTailSquared, &cMaxTailSquaredTot, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

MPI_Reduce( &cWeightedCycle, &cWeightedCycleTot, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
MPI_Reduce( &cWeightedCycleSquared, &cWeightedCycleSquaredTot, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

MPI_Reduce( &cWeightedTail, &cWeightedTailTot, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
MPI_Reduce( &cWeightedTailSquared, &cWeightedTailSquaredTot, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

if (mynode == 0)
{
    char res[20];
    sprintf(res, "results_%d.dat", n);
FILE * r = fopen(res, "w");

double ComponentsVariance = cComponentsSquaredTot - cComponentsTot*cComponentsTot;
double CyclicNodesVariance = cCyclicNodesSquaredTot - cCyclicNodesTot*cCyclicNodesTot;
double WeightedCycleVariance = cWeightedCycleSquaredTot - cWeightedCycleTot*cWeightedCycleTot;
double WeightedTailVariance = cWeightedTailSquaredTot - cWeightedTailTot*cWeightedTailTot;
double MaxCycleVariance = cMaxCycleSquaredTot - cMaxCycleTot*cMaxCycleTot;
double MaxTailVariance = cMaxTailSquaredTot - cMaxTailTot*cMaxTailTot

fprintf(r, "components: %lf \n", cComponentsTot);
fprintf(r, "components variance: %lf \n", ComponentsVariance);

fprintf(r, "cyclic nodes: %lf \n", cCyclicNodesTot);
fprintf(r, "cyclic nodes variance: %lf\n", CyclicNodesVariance);

fprintf(r, "avg cycle: %lf\n", cWeightedCycleTot);
fprintf(r, "avg cycle variance: %lf\n", WeightedCycleVariance);

fprintf(r, "avg tail: %lf\n", cWeightedTailTot);
fprintf(r, "avg tail variance: %lf\n", WeightedTailVariance);

fprintf(r, "image nodes: %lf\n", cImageNodesTot);
fprintf(r, "max cycle: %lf\n", cMaxCycleTot);
fprintf(r, "max cycle variance: %lf\n", MaxCycleVariance);

fprintf(r, "max tail: %lf\n", cMaxTailTot);
fprintf(r, "max tail variance: %lf\n", MaxTailVariance);

fclose(r);
}

void run()
{

MPI_Comm_size(MPI_COMM_WORLD, &totalnodes);
MPI_Comm_rank(MPI_COMM_WORLD, &mynode);

FILE * s;
if (mynode == 0)
{
    s = fopen(STATUS, "w");
    fprintf(s, "Allocating...
");
    fclose(s);
}
bool visit[n];
bool image[n];
/*maximum tail length for base [i]*/
int maxTAll[n];
/*maximum cycle length for base [i]*/
int maxCAll[n];
/*terminal nodes for base [i]*/
int terminalAll[i];

/*size of cycle for the component this node is a part of*/
int cycleSize[n];
/*distance to cycle from node n (0 if node n is cyclic)*/
int distToCycle[n];

/*The number of nodes -in- cycles of size [i] for current n*/
//int *allCResults = new int[n+1];
/*The number of nodes which are [i] away from their cycle*/
//int *allTResults = new int[n+1];
/*The number of tail nodes that lead to a cycle of size [i]*/
//int *allToCycleResults = new int[n+1];

/*Number of components for base i*/
int allComponents[n+1];
/*Number of image nodes for base i*/
int allCyclicNodes[n+1];
/*Sum of all nodes’ distance to cycle for base i*/
int allToCycleSum[n+1];
/*Sum of each node’s cycle length for base i*/
int allCycleLengthSum[n+1];

/*max cycle, max tail*/
int mC, mT;
int next, loc, baseTail, cycleLength, terminal;
int root, exp, base;

int listArray[n];
int listSize = 0;

if (mynode == 0)
{
}
s = fopen(STATUS, "w");
fprintf(s, "zeroing...\n");
fclose(s);
}

/*initialize arrays to 0 */
int i = 0;
for(i = 0; i < n; i++){
  if(i < n) {
    maxTAll[i] = 0;
    maxCAll[i] = 0;
    terminalAll[i] = 0;
  }
}
allComponents[n] = 0;
allCyclicNodes[n] = 0;
allToCycleSum[n] = 0;
allCycleLengthSum[n] = 0;

double t;
if (mynode == 0)
t = MPI_Wtime();

double tt;
double expTime = 0;
double tailTime = 0;
double intoCycleTime = 0;
double cycleTime = 0;
double resultsTime = 0;

if (mynode == 0)
{
  s = fopen(STATUS, "a");
  fprintf(s, "Prim root is %d...\n", root);
  fclose(s);
}

/*find the smallest primitive root*/
for(root = 1; !isPrimRoot(root); root++);

if (mynode == 0)
{
  s = fopen(STATUS, "a");
  fprintf(s, "Prim root is %d...\n", root);
  fclose(s);
}
int count = -1;

for(exp = 0; exp < n; exp++) {
    if(exp % 100 == 0 && mynode == 0) {
        s = fopen(STATUS, "w");
        fprintf(s, "Exp is %d\n", exp);
        fclose(s);
    }
    if(gcd(exp, n-1) != M_ARY) continue;
    count++;
    if(count % totalnodes != mynode) continue;

    base = m_exp(root, exp);
    mC = 0;
    mT = 0;
    /*0 out everything*/
    setArrays(cycleSize, visit, distToCycle, image);

    /*begin making graph, using gamma(i) = base^i mod n*/
    for(i = 1; i < n; i++) {
        if(visit[i]) continue;

        next = i;
        listArray[0] = next;
        listSize = 1;

        tt = MPI_Wtime();
        while(!visit[next]) {
            visit[next] = true;
            next = m_exp(base, next);
            image[next] = true;
            listArray[listSize] = next;
            listSize++;
        }
        expTime += MPI_Wtime() - tt;

        int j = 0;
        if(cycleSize[next] != 0) {
            if(distToCycle[next] == 0) {/*all tail into cycle*/
            tt = MPI_Wtime();
        }
cycleLength = cycleSize[listArray[listSize-1]];
if(listSize - 1 > mT) mT = listSize - 1;
for(j = 0; j < listSize-1; j++){
distToCycle[listArray[j]] = listSize - 1 - j;
cycleSize[listArray[j]] = cycleLength;
}
intoCycleTime += MPI_Wtime() - tt;
} else {/*extension of tail*/

   tt = MPI_Wtime();
   baseTail = distToCycle[listArray[listSize-1]];
   cycleLength = cycleSize[listArray[listSize-1]];
   if(listSize-1 + baseTail > mT) mT = listSize-1 + baseTail;
   for(j = 0; j < listSize-1; j++) {
      distToCycle[listArray[j]] = baseTail + listSize - 1 - j;
cycleSize[listArray[j]] = cycleLength;
   }
   tailTime += MPI_Wtime() - tt;
} }
else {/*new cycle found*/

   tt = MPI_Wtime();
   /*loc will be the first node in the cycle we ran in to*/
   int repeat = listArray[listSize-1];
   for(j = 0; listArray[j] != repeat; j++);

   int firstCycle = j;
cycleLength = listSize - (j+1);
   if(cycleLength > mC) mC = cycleLength;

   if(firstCycle > mT) mT = firstCycle;
   /*mark each tail node along the way with how far it is to*/
   /*the cycle (marked as a negative number)*/
   for(j = 0; j < firstCycle; j++) {
      distToCycle[listArray[j]] = firstCycle - j;
cycleSize[listArray[j]] = cycleLength;
   }
   /*mark each cycle nodes with how big the cycle is*/
   for(j = firstCycle; j < listSize - 1; j++)
cycleSize[listArray[j]] = cycleLength;

   allComponents[base]++;
   allCyclicNodes[base] += cycleLength;

   cycleTime += MPI_Wtime() - tt;
tt = MPI_Wtime();

terminal=0;
for(i = 1; i < n; i++)
    if(!image[i]) terminal++;
maxTAll[base] = mT;
maxCAll[base] = mC;
terminalAll[base] = terminal;
computeResults(distToCycle, cycleSize, &allToCycleSum[base], &allCycleLengthSum[base]);

resultsTime += MPI_Wtime() - tt;
if (mynode == 0)
{
    s = fopen(STATUS, "w");
    fprintf(s, "Writing Results...
");
    fclose(s);
}
writeTotalResults(
    maxTAll,
    maxCAll,
    terminalAll,
    allComponents,
    allCyclicNodes,
    allToCycleSum,
    allCycleLengthSum);

if (mynode == 0)
{
    s = fopen(STATUS, "w");
    fprintf(s, "%lf minutes...\n Exiting...\n%lf %lf %lf %lf\n", (MPI_Wtime() - t)/60, expTime, tailTime, intoCycleTime, cycleTime, resultsTime);
    fclose(s);
}
printf("%lf\n", expTime);
}
References


