#A67 INTEGERS 13 (2013)

COMPUTATION OF AN IMPROVED LOWER BOUND TO GIUGA'S PRIMALITY CONJECTURE

Jonathan Borwein
Centre for Computer-assisted Research Mathematics and its Applications (CARMA), School of Mathematical and Physical Sciences, University of Newcastle, Australia
jonathan.borwein@newcastle.edu.au jborwein@gmail.com

Christopher Maitland
Mathematical Sciences Institute, Australian National University, Canberra, Australia
c3053540@gmail.com

Matthew Skerritt
Centre for Computer-assisted Research Mathematics and its Applications (CARMA), School of Mathematical and Physical Sciences, University of Newcastle, Australia
matt.skerritt@gmail.com

Received: 3/7/13, Accepted: 9/5/13, Published: 10/10/13

Abstract

Our most recent computations tell us that any counterexample to Giuga's 1950 primality conjecture must have at least 19,908 decimal digits. Equivalently, any number which is both a Giuga and a Carmichael number must have at least 19,908 decimal digits. This bound has not been achieved through exhaustive testing of all numbers with up to 19,908 decimal digits, but rather through exploitation of the properties of Giuga and Carmichael numbers. This bound improves upon the 1996 bound of one of the authors. We present the algorithm used, and our improved bound. We also discuss the changes over the intervening years as well as the challenges to further computation.

1. Introduction

In a 1950 paper [9] Giuseppe Giuga formulated his now well-known prime number conjecture:

Conjecture 1 (Giuga Conjecture)**.** An integer, $n > 1$, is prime if and only if

$$s_n := \sum_{k=1}^{n-1} k^{n-1} \equiv -1 \pmod{n}.$$

In 1995, Takahashi Agoh [1] showed that this is equivalent to:

Conjecture 2 (Giuga-Agoh Conjecture)**.** An odd integer, $n > 1$, is prime if and only if the even Bernoulli number B_{n-1} satisfies

$$nB_{n-1} \equiv -1 \pmod{n}.$$

A more recent generalisation of Giuga's conjecture can be found in [12].

1.1. Giuga and Carmichael numbers

Herein, we present enough background material to facilitate understanding of the algorithm with which lower bounds to any possible counterexample were computed. For more details, including proofs of the theorems below, see [6, 7, 8].

We begin by observing that if p is prime, then $s_p \equiv p-1 \pmod{p}$ is an immediate consequence of Fermat's little theorem. The question, then, is whether there exists any composite n with $s_n \equiv n-1 \pmod{n}$. To this end, Giuga proved the following theorem.

Theorem 3. *An integer, $n > 1$, satisfies $s_n \equiv -1 \pmod{n}$ if and only if each prime divisor p of n satisfies:*

$$p \mid \left(\frac{n}{p} - 1\right), \tag{1}$$

$$(p-1) \mid \left(\frac{n}{p} - 1\right). \tag{2}$$

□

Any composite number satisfying (1) is known as a *Giuga number.* It is currently unknown whether there are infinitely many Giuga numbers, or whether there are any odd Giuga numbers. Recent papers on Giuga numbers include [10, 11]. Note that Giuga numbers must be square-free. To see this we suppose $n = p^2q$ and observe that (1) implies $p \mid (pq-1)$, which is impossible.

Giuga numbers may be characterised in the following manner, which is useful in the computation of exclusion bounds for counterexamples.

Theorem 4. *A composite number $n \in \mathbb{N}$ is a Giuga number if and only if it is square-free and satisfies*

$$\sum_{p \mid n} \frac{1}{p} - \prod_{p \mid n} \frac{1}{p} \in \mathbb{N}.$$

□

Composite, square-free numbers satisfying (2) are called *Carmichael numbers.* We may replace (2) with the equivalent formulation:

$$(p-1) \mid (n-1). \tag{3}$$

It has been shown that there are infinitely many Carmichael numbers [2]. Indeed, for any $\varepsilon > 0$, for sufficiently large x the number of Carmichael numbers $n \leq x$ exceeds $x^{\beta-\varepsilon}$ where $\beta := 0.33222... > 2/7$ is an explicit constant. More recently, various interesting density estimates have been given in [5, 14, 16]. For example, the number of Carmichael numbers $n \leq x$ that are not Giuga numbers exceeds $x^{\beta-\varepsilon}$ for sufficiently large x. It has been observed that $x^{\beta-\varepsilon}$ may be replaced by $x^{1/3}$ as a consequence of a result of Harman [13].

1.2. Normal sequences of primes

Note that if p is a prime divisor of a Carmichael number, n say, then for no $k \in \mathbb{N}$ is $kp+1$ a prime divisor of n. We can see this if we suppose that there is some prime p such that $p \mid n$, $kp+1$ is prime, and $(kp+1) \mid n$. Then, by (3), $(kp+1)-1 = kp \mid (n-1)$ which is impossible if $p \mid n$. A consequence of this observation is that $p = 2$ cannot be a prime divisor of a Carmichael number, and that therefore every Carmichael number must be odd.

We say that the prime divisors of a Carmichael number, n, are *normal* according to the following definition:

Definition 5. A set of odd primes, P, is *normal* if no $p, q \in P$ satisfies $p \mid (q-1)$.

The following extension of the idea of a normal set of primes is useful in the computation of exclusion bounds.

Definition 6. Let P be a normal set of primes. For any prime, p, we say p *is normal to* P if $P \cup \{p\}$ is normal.

In light of this we can reformulate Theorem 3 to say that a counterexample to Giuga's conjecture must be both a Giuga number and a Carmichael number. Giuga's conjecture then simply states that there is no such number. A detailed study on Giuga's conjecture and variants related to normal families of primes is made in [17].

2. Exclusion Bounds

We may compute exclusion bounds for counterexamples to Giuga's conjecture by exploiting the properties of Giuga and Carmichael numbers described in the previous section.

Notation 1 (Odd primes and sets of odd primes)**.**

- Let q_k denote the kth odd prime.
- Let Q denote the collection of all odd primes $\{3, 5, 7, 11, \ldots\}$

- Let Q_k denote the set $\{q_1, \ldots, q_{k-1}\}$ of odd primes strictly less than q_k
- Let $\mathcal{N}_d$ denote the set $\{\mathcal{N} \subset Q_d \mid \mathcal{N} \text{ is normal}\}$ of all finite normal sets whose maximal element is less than q_d.

To begin with, an easy lower bound can be found by observing that, by Theorem 4, the reciprocal sum of prime factors of any Giuga number must be greater than 1, and that any Carmichael number must be odd. We observe that the set $Q_{10} = \{3, 5, 7, 11, 13, 17, 19, 23, 29\}$ is the smallest set of odd primes whose reciprocal sum is greater than 1. Thus a counterexample must, because $|Q_{10}| = 9$, have at least 9 prime factors and, because $\prod_{p \in Q_{10}} p > 10^9$, must have at least 10 digits.

Observe that the above result does not fully take into account the fact that a counterexample must also be a Carmichael number. Doing so raises the bounds considerably.

We exploit a binary tree structure, where the vertices of the tree are normal sets of primes. We begin with the empty set as the root of the tree, and consider this to be depth 1 of the tree. Each vertex, $\mathcal{N}$ say, at depth d of the tree has itself as a child. Additionally, if q_d is normal to $\mathcal{N}$ then $\mathcal{N} \cup \{q_d\}$ is also a child of $\mathcal{N}$. This structure is shown in Figure 1. With this structure, the collection of all sets at depth d of this tree is precisely the set $\mathcal{N}_d$, defined above.

$$\begin{tikzcd}
 & & & & \{\} \arrow[dll] \arrow[drrr] & & & & \\
 & & \{\} \arrow[dl] \arrow[drrr] & & & & & \{3\} \arrow[dl] \arrow[dr] & \\
 & \{\} \arrow[dl] \arrow[dr] & & & \{5\} \arrow[d] \arrow[dr] & & \{3\} \arrow[d] & & \{3,5\} \arrow[d] \\
\{\} \arrow[d] \arrow[dr] & & \{7\} \arrow[d] \arrow[dr] & & \{5\} \arrow[d] & \{5,7\} \arrow[d] & \{3\} \arrow[d] \arrow[dr] & & \{3,5\} \arrow[d] \\
\{\} & \{11\} & \{7\} & \{7,11\} & \{5\} & \{5,7\} & \{3\} & \{3,11\} & \{3,5\}
\end{tikzcd}$$

Figure 1: The first 5 levels of a tree of normal sets of primes.

For each set, $\mathcal{N}$, on level d of the tree we compute a new set which we call $T_d(\mathcal{N})$. We begin by assigning $T_d(\mathcal{N}) = \mathcal{N}$. We then look at the reciprocal sum of $T_d(\mathcal{N})$

$$r := \sum_{q \in T_d(\mathcal{N})} \frac{1}{q}$$

If $r \leq 1$ and $\mathcal{N} \cup \{q_d\}$ is normal then we set $T_d(\mathcal{N}) = T_d(\mathcal{N}) \cup \{q_d\}$, otherwise we leave it unchanged. We repeat this process with q_{d+1}, q_{d+2} and so on until $r > 1$, at which point we are done.

Definition 7 (j-value)**.** Let $\mathcal{N} \in \mathcal{N}_d$ be a vertex at depth d of the tree. Denote by $T_d(\mathcal{N})$ the set computed as described, above. Then we say that

$$j_d(\mathcal{N}) := |T_d(\mathcal{N})|$$

is the *j-value* or, equivalently, the *score* of $\mathcal{N}$.

Note that there is a special case when the reciprocal sum of $\mathcal{N}$ is greater than 1 to begin with. In this case, no primes are appended to $T_d(\mathcal{N})$, and we have $T_d(\mathcal{N}) = \mathcal{N}$ and so $j_d(\mathcal{N}) = |\mathcal{N}|$.

Example 8. The lower bound we computed at the beginning of this section is equivalent to computing $T_1(\{\}) = \{3, 5, 7, 11, 13, 17, 19, 23, 29\}$ and $j_1(\{\}) = 9$. We may also compute

$$T_2(\{\}) = \{5, 7, 11, 13, 17, \ldots, 107, 109\} \text{ so that } j_2(\{\}) = 27$$
$$T_2(\{3\}) = \{3, 5, 11, 17, 23, \ldots, 317, 347\} \text{ so that } j_2(\{3\}) = 32.$$

Observe that, in general, it is not true that $T_d(\mathcal{N})$ is a normal set; our criterion for inclusion of a prime, q, into $T_d(\mathcal{N})$ is that it is normal to $\mathcal{N}$, and not that it is normal to $T_d(\mathcal{N})$. We can see, above, that both $T_2(\{\})$ and $T_2(\{3\})$ contain both 5 an 11, and so are not normal. □

For a fixed depth, d, the values of $j_d(\mathcal{N})$ for all $\mathcal{N} \in \mathcal{N}_d$ allow us to compute a lower bound for the number of primes needed for a counterexample. For each $\mathcal{N}$, we consider a hypothetical family of all counterexamples whose set of prime factors less than q_d is precisely $\mathcal{N}$, and $j_d(\mathcal{N})$ is an approximation—specifically a lower bound—of the number of primes required for any such counterexample, should one exist.

To see that $j_d(\mathcal{N})$ is a lower bound, recall that $T_d(\mathcal{N})$ is not (in general) a normal set, but that the prime factors of a counterexample must form a normal set. As such, whenever $T_d(\mathcal{N})$ is not normal, some of its elements must be absent in any counterexample from the above hypothetical family. Each such absent element—because of the way $T_d(\mathcal{N})$ was constructed—must be replaced by at least one prime greater than $\max(T_d(\mathcal{N}))$ in order for the reciprocal sum to stay above 1. Note that if $T_d(\mathcal{N})$ is normal, then we have the smallest normal set containing $\mathcal{N}$ with a reciprocal sum greater than 1, although it is not necessarily the case that these normal primes are the factors of a counterexample. We may, nonetheless, conclude that $j_d(\mathcal{N})$ is a lower bound on the number of prime factors in any counterexample whose prime factors less than q_d are exactly the set $\mathcal{N}$.

Recall that $\mathcal{N}_d$ is the collection of all normal subsets of Q_d and that the leaves (or vertices) of our tree for a fixed depth, d, are precisely the sets contained in $\mathcal{N}_d$.

Any counterexample to Giuga's conjecture must have as its set of prime factors less than q_d exactly one of $\mathcal{N} \in \mathcal{N}_d$, and so if we compute $j_d(\mathcal{N})$ for all $\mathcal{N} \in \mathcal{N}_d$, then the smallest such value

$$j_d := \min \{j_d(\mathcal{N}) \mid N \in \mathcal{N}_d\}$$

must be a lower bound for the number of prime factors in any counterexample to Giuga's conjecture. From this we compute the corresponding lower bound

$$P_d := \prod_{k=1}^{j_d} q_k$$

as the product of the first j_d odd primes.

Example 9. Any counterexample to Giuga's conjecture will either have 3 as a prime factor or not, corresponding to the normal sets $\{\}$ and $\{3\}$ sitting at depth 2 of the tree in Figure 1. We have computed the values of $j_2(\{\})$ and $j_2(\{3\})$ in Example 8, above. In the former case there must be at least 32 prime factors; in the latter case there must be at least 27. We may conclude, then, that such a counterexample must have at least 27 prime factors—that is $j_2 = 27$—and must therefore be greater than $P_2 = \prod_{k=1}^{27} q_k > 10^{42}$. □

The algorithm, then, is simply to build the tree down to the desired depth, d, then to calculate the j-value for every vertex at that depth, so as to find the minimum j-value as the value of j_d. P_d may easily be computed once j_d is known. Regrettably the number of vertices grows exponentially with the depth of the tree, so computation of j_d for large d is slow and so the algorithm is of limited use in the form hitherto described. Fortunately a relatively simple observation leads to significant improvement in performance.

We observe that the j-values of a set in our tree do not decrease as we travel down the tree. That is, for a set $\mathcal{N}$ at depth d of the tree, the value of $j_d(\mathcal{N})$ is no larger than the value of $j_{d+1}(\mathcal{N}')$ so long as $\mathcal{N}'$ is a child of $\mathcal{N}$. This follows from the following theorem which is proved in [8].

Theorem 10. *Let $\mathcal{N} \in \mathcal{N}_d$ be a normal set of primes. Then $j_{d+1}(\mathcal{N}) \geq j_d(\mathcal{N})$ Furthermore, let $\mathcal{N}' = N \cup \{q_d\}$. If $\mathcal{N}'$ is normal, then $j_{d+1}(\mathcal{N}') \geq j_d(\mathcal{N})$.* □

A corollary of this theorem is that any normal superset of $\mathcal{N}$ whose reciprocal sum is greater than 1 must have at least $j_d(\mathcal{N})$ elements. That is, if $\mathcal{N} \subset N'$, $\mathcal{N}'$ is normal, and $\sum_{q \in \mathcal{N}'} 1/q > 1$, then $|\mathcal{N}'| \geq j_d(\mathcal{N})$.

We exploit Theorem 10 to significantly cut down the number of vertices (leaves) needed when computing j_d for a fixed d. Inasmuch as j_d is the minimum j-value of any vertex at depth d of the tree, the j-value of any vertex at that depth is an upper bound for j_d. Any vertex higher in the tree—which is to say at depth $d' < d$—whose j-value exceeds such an upper bound need not have any of its children examined.

$$\begin{tikzcd}
& & & & & & & \{\}_{9} \arrow[dlll] \arrow[drrr] & & & & \\
& & & & \{\}_{27} \arrow[dlll] \arrow[drr] & & & & & & \{3\}_{36} \arrow[dl] \arrow[dr] & \\
& \{\}_{66} \arrow[dl] \arrow[drr] & & & & & \{5\}_{65} \arrow[dl] \arrow[drr] & & & \{3\}_{396} & & \{3,5\}_{383} \\
\{\}_{144} & & & \{7\}_{114} \arrow[dl] \arrow[dr] & & \{5\}_{196} & & & \{5,7\}_{127} \arrow[d] & & & \\
& & \{7\}_{221} & & \{7,11\}_{216} & & & & \{5,7\}_{127} & & &
\end{tikzcd}$$

Figure 2: The first 5 levels of the trimmed tree of normal sets of primes.

In essence, we may trim the tree early. A trimmed tree, along with the j-values for each vertex can be seen in Figure 2.

Note that the algorithm still exhibits signs of exponential growth (see below), however the work saved by this method is enough to make the algorithm much more tractable and proved sufficient, in 1996, to make the difference between being able to calculate j_{19} and j_{100} (see [8]).

The amount of work saved by the above optimisation is entirely dependent on the choice of initial upper bound. An excellent candidate for a good such bound is given by the family of sets described as follows:

Definition 11 (L-sets)**.** Let $k \geq 5$ then

$$L_5 := \{5, 7\} \quad \text{and} \quad L_{k+1} := \begin{cases} L_k \cup q_k & \text{if this set is normal,} \\ L_k & \text{otherwise.} \end{cases}$$

In every computation we have performed for depths $d > 5$ the value of j_d has been given by these sets. That is to say that $j_d = j_d(L_d)$ for all $5 \leq d \leq 311$.

These L-sets lead to an interesting observation: the algorithm described here is exhaustible. The first of these sets to have a reciprocal sum greater than 1 is the set $L_{27{,}692}$, which has 8,135 elements. Note that the primes in $L_{27{,}692}$ are not the factors of a counterexample to Giuga's conjecture. Nonetheless, it must be the case that $j_d \leq$ 8,135 for all $d \geq$ 27,692, and so our algorithm can never compute an exclusion bound higher than 8,135 prime factors, even if it turns out that $j_{27{,}692} < j_{27{,}692}(L_{27{,}692})$. It is the sincerest hope of the authors to eventually exhaust this algorithm, although given the apparently exponential growth of even the trimmed tree, it is exceptionally unlikely to happen.

It was with this algorithm in 1996 that Borwein et al [6] were able to compute $j_{135} = 3{,}459$. Similarly, it is with this algorithm that the present authors have been able to compute $j_{311} = 4{,}771$ thus obtaining the following result.

Theorem 12. *Any counterexample to Giuga's primality conjecture is an odd square-free number with at least* 4,771 *prime factors and so must exceed* $10^{19{,}907}$. □

3. Computation Improvements, Implementation Details, and Technical Challenges

In 1996, Borwein et al [6] were able to compute j_{100} in *Maple* using the algorithm described above. Borwein reports in [8] that the computation required "a few CPU hours for each k" for values of k near 100. Using a C++ implementation (which took two months to produce) they were able to compute j_{135}, taking "303 CPU hours" [6]. The C++ implementation then crashed irrevocably before doing any extra work [8]. Thus, in 1996, the best results for exclusion bounds for Giuga's conjecture were

$$j_{135} = 3{,}459 \qquad\qquad P_{135} > 10^{13{,}886}$$

No further work on these computations was done by any of the present authors until mid-2009. Effort was spent to reduce the number of operations in the implementation, without deviating from the algorithm. In early 2011, j_{173} was computed in approximately thirty five hours using *Maple*, obtaining:

$$j_{173} = 3{,}795 \qquad\qquad P_{173} > 10^{15{,}408}$$

The main issue with the *Maple* code was long run times, although memory issues were beginning to creep in. An attempt was made to parallelise the algorithm using an early version of *Maple*'s inbuilt multithreading capabilities. At the time, *Maple* 13 was the current version. Unfortunately this attempt failed due to poor machine memory management, which was most likely a result of the infancy of the threading library.

Significant execution time improvements were achieved with a C++ implementation. In contrast to the authors' earlier experience with C++ (described above, and in [8]) the implementation was now easy and straightforward, took only a day or two, and worked extremely well. This implementation was able to compute j_{173} in a little under ten minutes. This implementation stored only the leaves of the tree and in the interests of simplicity of code, did not store the tree after the computation had completed.

Remark 13. We note here that a second, equivalent approach to exploiting Theorem 10 was experimented with at this time. The variant is to build the tree up from the root vertex {} by iteratively computing only the immediate children of the vertex with the lowest j-value, favoring vertices closer to the root in the case of a tie. This variation has the advantage that it always processes the least number of vertices possible in any execution. In practice, it showed small (but essentially insignificant) speed improvements over the method described above, most likely due to the exceptionally good bounds provided by the L_d sets. Unfortunately, this variation did not easily lend itself to a threaded (parallel) implementation and has not been used since.

Further execution time improvements were achieved by threading the code. Threading was achieved via use of Apple's threading library "Grand Central Dispatch", which conceals the traditional low-level tools of threading (semaphores, threads, mutexes, etc) and instead presents a much more intuitive method of passing code blocks into work queues, allowing the library to do the rest (see [3, 4] for more information). This allowed for a quick and painless threading implementation and avoided nearly all of the hurdles normally associated with introducing threading to an existing program. It took approximately two days to implement the threading, and this new threaded implementation could compute j_{173} in approximately two minutes.

A grid implementation, running on an Apple X-grid cluster, was experimented with. Unfortunately the grid facilities lacked any easy interprocess communications and the problem did not divide into evenly sized sub-problems. This avenue was abandoned in favor of improving the already working single-machine threaded implementation.

3.1. Final computations

With this implementation we were able to compute to depth 280 and obtained the following results.

$$j_{280} = 4{,}581 \qquad\qquad P_{280} > 10^{19{,}023}$$

During the computation to depth 280 our implementation was exceeding the memory of the computer it was running on. Computation speed drastically slowed due to excessive paging. Unfortunately, no detailed records were kept of this computation, however we know the machine had 12 or 14 gigabytes of memory. To fix the paging problem, it was realised that much of the data in memory consisted of leaves which had completed processing. The program was thus modified to write these completed leaves to disk, in the form of a Berkeley database file, to save on RAM. The Berkeley database format was chosen because it already incorporated caching, as well as sorting of the entries (see [15] for more information). The newly modified program was run and computed to depth 301 to obtain the following results:

$$j_{301} = 4{,}669 \qquad\qquad P_{301} > 10^{19{,}432}$$

The computation took approximately ninety five hours.

A side effect of the change was that, since the completed tree was now being written to disk, later computations could pick up where previous computations had left off[1]. Unfortunately, similar memory exhaustion and excessive paging problems had crept back in by the end of the computation. To fix this problem, the program was modified to write all leaves to disk and to only hold in memory those leaves which were currently being processed.

The new implementation was able to compute $j_{311} = 4{,}771$, taking approximately an extra six days[2] and producing an output file of approximately one hundred and twenty gigabytes.

$$j_{311} = 4{,}771 \qquad\qquad P_{311} > 10^{19{,}907}$$

Memory issues are still present, although the most recent implementation delayed them until approximately depth 308. The most recent memory issues appear to be caused not by large amounts of tree data, but by large numbers of jobs waiting to be executed by the threading API[3]. If this is the case, then we begin to see a cost of the easy implementation of threading thanks to the Grand Central Dispatch library. It may be the case, however, that the memory issues are caused by tree data being created faster than the existing data can be written to disk, combined with the serialised writing to disk. Efforts to identify and eradicate the memory problems are ongoing, and no further computations have yet been performed.

We should note the diminishing returns of these fixes to memory exhaustion; each fix has taken longer to implement and has yielded less improvement than the fix which preceded it. Growth of execution time and file output size is discussed in the next section.

4. Runtime Analysis

Due to the unpredictable nature of the placement of primes in general, and of normal pairs of primes in particular, it is very difficult to analyse the runtime and space complexity of the above algorithm algebraically. However, measurements of runtimes and file output sizes strongly suggest the trimmed tree still grows exponentially.

[1] Unfortunately, since this was the first time this implementation was being run, it was unable to exploit this feature and the quoted ninety five hours was to compute the tree in its entirety to depth 301.

[2] That is, the computation which started at depth 301 from the output of the previous implementation and ran through to depth 311 took six days. This brings the total execution time for depth 311 to over ten days.

[3] Tree data is stored on disk and not passed into processing functions. Instead, the first act of the function which processes the leaf is to retrieve the leaf from disk.

The following measurements were taken on a 2.8Ghz quad core i7 processor with 16 gigabytes of RAM, running Mac OS X 10.7 (Lion). The computations were performed by the most recent implementation as described in the previous section—the one used to compute j_{311}. The program computed j_d for all depths $1 \leq d \leq 311$. The ability of this implementation to begin a new computation starting from a previously completed computation was exploited to measure the size of the output file of the tree at each depth. Total computation time up to depth d was calculated by adding together the times for all computations from depth d' to $d' + 1$ for $d' < d$.

The granularity of the time measurements is 1 second and as such the total computation times for small d may be a little unreliable, as the sub-second execution times are rounded to 0 or 1 seconds. This should not affect our conclusions very much when we consider that the individual execution times reach into the minutes at around depth $d = 66$, hours at around depth $d = 95$, and eventually even days.

Of more concern is the timings for individual executions of depths $d > 301$. The computer these computations were performed on was the personal computer of one of the authors, and with the execution of a single step in depth taking close to a day (or more for the later ones), it was sometimes necessary to pause the computation for hours at a time to allow the computer to be used for other purposes. It is unclear whether these pauses were included in the timings or not. As such the timings for those computations are somewhat suspect and have not been used in any of the following analysis. Note that these pauses will not have affected file output sizes and so the file size data for these computations has been kept and used.

Note that for any $\mathcal{N} \in \mathcal{N}_d$ for which $\mathcal{N} \cup \{q_d\}$ is not normal, $\mathcal{N}$ only has one child in the tree (namely, itself), and $j_{d+1}(\mathcal{N}) = j_d(\mathcal{N})$ as a direct result the method by which the j-values are computed. So in the case that the vertex, $\mathcal{N}$, at depth d whose j-value realises the value j_d[4] (i.e., $\mathcal{N} \subset \mathcal{N}_d$ where $j_d(\mathcal{N}) = j_d$), if q_d is not normal to $\mathcal{N}$ we have that $j_{d+1} = j_d$ and no other computation for depth $d + 1$ needs to be performed. The program was modified to recognise this condition and to simply skip ahead to the next depth, $d' > d$ for which $\mathcal{N} \cup \{q_{d'}\}$ is normal. As such the data appears to "miss" some depths, but this is simply the program skipping these trivial tree depths.

When we plot the log of total execution time, or the log of output file size, against depth (see Figure 3) we see an essentially linear pattern, consistent with exponential growth. We expect fluctuation in the number of vertices processed due to the unpredictable nature of which primes will and will not be included in the computation of $j_d(\mathcal{N})$ for any given vertex $\mathcal{N}$, so the observed variation from linear behavior is not inconsistent. Furthermore, the particulars and vagaries of the output file format (a Berkeley database) might also be a source of some variation

[4]In principle there is nothing in the algorithm to prevent multiple sets realising the minimum j-value, but in practice this has not yet been observed.

in the file sizes.

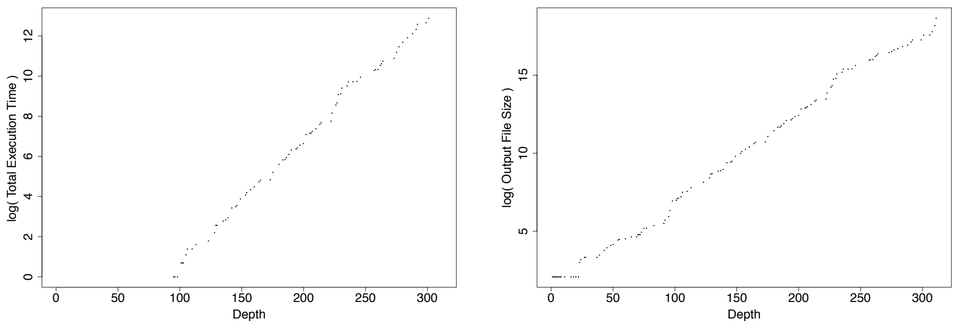

Figure 3: Log-linear plot of cumulative execution time in seconds (left) and output file size in kilobytes (right) against tree depth.

4.1. Predicted future performance

Applying a least-squares linear model to this data, we make the following 95% prediction intervals, for the current implementation of the algorithm running on the machine described, above. Total computation time for j_{350} is expected to take between 69 and 188 days and is expected to output a data file of between 490 gigabytes and 2.2 terabytes. We expect the total computation time for j_{400} to be between 4 and 12 years and expect an output file between 7.5 and 36 terabytes in size.

In addition, the following predictions are found by looking at the 95% prediction intervals for predictions of all depths 312 through to 400, and choosing those depths which contain the desired size or time respectively. We predict a terabyte sized output file at a depth between 337 and 364, and we predict a cumulative month long computation will be needed for depths between 321 and 337, and a cumulative year long computation for depths between 361 and 377.

5. Further Work

The current bottleneck in computations appears to be storage; both RAM and hard disk. However, the statistical predictions suggest that time will shortly become the bottleneck for further computations. It is the authors' intention to fix the RAM issues and to compress the output file to address this growth. It is hoped that doing so will delay memory growth enough to once again make execution time the bottleneck . It is also the authors' intention to re-explore grid computation, most probably with the use of the Open MPI API in the hopes that spreading the load across many computers will help alleviate both the time and the storage limitations.

Acknowledgements

The authors would like to thank Dr David Allingham for his support and advice regarding the CARMA computational infrastructure, and Dr Robert King for his advice in interpreting the statistical results.

References

[1] Takashi Agoh. On Giuga's conjecture. *Manuscripta Math.*, 87(4):501–510, 1995.

[2] W. R. Alford, Andrew Granville, and Carl Pomerance. There are infinitely many Carmichael numbers. *Ann. of Math. (2)*, 139(3):703–722, 1994.

[3] Apple Inc. Concurrency programming guide. `http://developer.apple.com/library/ios/#documentation/General/Conceptual/ConcurrencyProgrammingGuide/`, 2013. [Online; accessed 25-Jan-2013].

[4] Apple Inc. Grand central dispatch (gcd) reference. `http://developer.apple.com/library/ios/#documentation/Performance/Reference/GCD_libdispatch_Ref/`, 2013. [Online; accessed 25-Jan-2013].

[5] William D. Banks, C. Wesley Nevans, and Carl Pomerance. A remark on Giuga's conjecture and Lehmer's totient problem. *Albanian J. Math.*, 3(2):81–85, 2009.

[6] D. Borwein, J. M. Borwein, P. B. Borwein, and R. Girgensohn. Giuga's conjecture on primality. *Amer. Math. Monthly*, 103(1):40–50, 1996.

[7] J. M. Borwein and E. Wong. A survey of results relating to Giuga's conjecture on primality. In *Advances in mathematical sciences: CRM's 25 years (Montreal, PQ, 1994)*, volume 11 of *CRM Proc. Lecture Notes*, pages 13–27. Amer. Math. Soc., Providence, RI, 1997.

[8] Jonathan M Borwein, David H Bailey, and Roland Girgensohn. *Experimentation in Mathematics*. Computational Paths to Discovery. A K Peters Ltd, 2004.

[9] Giuseppe Giuga. Su una presumibile proprietà caratteristica dei numeri primi. *Ist. Lombardo Sci. Lett. Rend. Cl. Sci. Mat. Nat. (3)*, 14(83):511–528, 1950.

[10] José María Grau, Florian Luca, and Antonio M. Oller-Marcén. On a variant of Giuga numbers. *Acta Math. Sin. (Engl. Ser.)*, 28(4):653–660, 2012.

[11] José María Grau and Antonio M. Oller-Marcén. Giuga numbers and the arithmetic derivative. *J. Integer Seq.*, 15(4):Article 12.4.1, 5, 2012.

[12] Jos María Grau and Antonio M. Oller-Marcén. Generalizing Giuga's conjecture. *ArXiv e-prints*, March 2011.

[13] Glyn Harman. Watt's mean value theorem and Carmichael numbers. *Int. J. Number Theory*, 4(2):241–248, 2008.

[14] Florian Luca, Carl Pomerance, and Igor Shparlinski. On Giuga numbers. *Int. J. Mod. Math.*, 4(1):13–18, 2009.

[15] Oracle. Oracle berkeley db. `http://www.oracle.com/technetwork/products/berkeleydb/overview/index.html`, 2013. [Online; accessed 25-Jan-2013].

[16] Vicentiu Tipu. A note on Giuga's conjecture. *Canad. Math. Bull.*, 50(1):158–160, 2007.

[17] E. Wong. Computations on normal families of primes. *MSc, Simon Fraser University*, 1997.