AN ALGORITHM AND ESTIMATES FOR THE ERDŐS-SELFRIDGE FUNCTION (WORK IN PROGRESS)

BRIANNA SORENSON, JONATHAN P. SORENSON, AND JONATHAN WEBSTER

Abstract. Let $p(n)$ denote the smallest prime divisor of the integer n. Define the function $g(k)$ to be the smallest integer $> k+1$ such that $p(\binom{g(k)}{k}) > k$. So we have $g(2) = 6$ and $g(3) = g(4) = 7$.

In this paper we present the following new results on the Erdős-Selfridge function $g(k)$:

(1) We present a new algorithm to compute the value of $g(k)$, and use it to both verify previous work [1, 16, 12] and compute new values of $g(k)$, with our current limit being

$$g(323) = 1\ 69829\ 77104\ 46041\ 21145\ 63251\ 22499.$$

(2) We define a new function $\hat{g}(k)$, and under the assumption of our *uniform distribution heuristic* we show that

$$\log g(k) = \log \hat{g}(k) + O(\log k)$$

with high "probability". We also provide computational evidence to support our claim that $\hat{g}(k)$ estimates $g(k)$ reasonably well in practice.

(3) There are several open conjectures on the behavior of $g(k)$ from [1] which we are able to prove for $\hat{g}(k)$, namely that for constants $c_1 = 0.525\ldots$ and $c_2 = 1$,

$$c_1 + o(1) \quad \leq \quad \frac{\log \hat{g}(k)}{k/\log k} \quad \leq \quad c_2 + o(1),$$

and that

$$\limsup_{k\to\infty} \frac{\hat{g}(k+1)}{\hat{g}(k)} = \infty.$$

(4) Let $G(x,k)$ count the number of integers $n \leq x$ such that $p(\binom{n}{k}) > k$. Unconditionally, we prove that for large x, $G(x,k)$ is asymptotic to $x/\hat{g}(k)$.

(5) And finally, we show that the running time of our new algorithm is at most $g(k)\exp[-c(k\log\log k)/(\log k)^2(1+o(1))]$ for a constant $c > 0$. Note that our algorithm deals with two sub-problems that have both been proven to be NP-complete: the knapsack problem [4] and finding the smallest solution to a system of modular congruences [13].

For previous work on the Erdős-Selfridge function, see [1, 2, 16, 12, 11, 5].

1. Introduction

As stated in the abstract above, let $p(n)$ denote the smallest prime divisor of the integer n, and define the function $g(k)$ to be the smallest integer $> k+1$ such that $p(\binom{g(k)}{k}) > k$. So we have $g(2) = 6$ and $g(3) = g(4) = 7$.

We begin with a discussion of previous work on $g(k)$, then state our new results, and finally outline the rest of this paper.

1.1. Previous Work.

Paul Erdős introduced the problem of estimating the function $g(k)$ in 1969 [3]. He, along with Ecklund and Selfridge [1] showed that $g(k) > k^{1+c}$ for a small constant c, showed that $g(k) < e^{k(1+o(1))}$, and tabulated $g(k)$ up to $k = 40$, plus $g(42)$, $g(46)$, and $g(52)$. They also stated several conjectures on the behavior of $g(k)$:

(1) $\limsup_{k\to\infty} \frac{g(k+1)}{g(k)} = \infty$,
(2) $\liminf_{k\to\infty} \frac{g(k+1)}{g(k)} = 0$,
(3) $g(k)$ is super-polynomial in k,
(4) $\lim_{k\to\infty} g(k)^{1/k} = 1$,
(5) and that $g(k) < \exp[ck/\log k]$ for a constant $c > 0$.

So far, only (3) has been proven. Note that (5) implies (4).

Scheidler and Williams [16] described how to use Kummer's theorem to construct a sieving problem to compute $g(k)$, and they proceeded to find $g(k)$ for all $k \le 140$ (they have a typo: $g(114) = 59819\ 90286\ 02614$). Kummer's theorem states that a prime p does not divide $\binom{n}{k}$ if and only if the digits of n's representation in base p match or exceed the corresponding digits of k's representation in base p. Let $M_k = \prod_{p\le k} p^{\lfloor \log_p k\rfloor+1}$. This theorem allows one to set up a sieve problem to search for $g(k)$ as the smallest residue, larger than $k+1$, modulo M_k, that satisfies Kummer's criteria. Lukes, Scheidler, and Williams [12] then improved their sieve, used special-purpose hardware, and computed $g(k)$ for all $k \le 200$.

A complete table of previously known values of $g(k)$ is available online from the *Online Encyclopedia of Integer Sequences* (A003458) at `https://oeis.org/A003458`.

Erdős, Lacampagne, and Selfridge [2] showed that

$$g(k) \gg k^2/\log k,$$

improving the lower bound stated above. Granville and Ramaré [5] improved this to

$$g(k) > k^{c\sqrt{\log k/\log\log k}}$$

for a constant $c > 0$, thereby proving conjecture (3). Konyagin [11] improved this even further to

$$g(k) > k^{c\log k}$$

for a constant $c > 0$.

See also [6, §B31]. As far as we are aware, no further results on $g(k)$ have been published since 1999.

1.2. **New Results.** We adapted the sieving techniques from [16, 12] to use the space-saving wheel sieve, which was described in [17], and was used previously to find pseudosquares [18], pseudoprimes [19], and primes in patterns [20]. Our resulting algorithm has, so far, verified all previous computations for $g(k)$, and extended them for all $k \leq 323$. Full tables of results appear later, but we have

$$g(323) = 1\ 69829\ 77104\ 46041\ 21145\ 63251\ 22499.$$

Values of $g(k)$ for $k \leq 272$ were found using a single processor core. Subsequent values were found using a cluster of 192 cores.

Our analysis makes use of a *uniform distribution heuristic.* Recall that $M_k := \prod_{p \leq k} p^{\lfloor \log_p k \rfloor + 1}$. If we let R_k denote the number of acceptible residues, under Kummer's theorem, modulo M_k, and if these residues are, in a sense, uniformly distributed up to M_k, then we expect $g(k)$ to be roughly M_k/R_k. In fact, we define

$$\hat{g}(k) := M_k/R_k.$$

Under the assuption of our uniform distribution heuristic, we prove that, with high probability,

$$\log g(k) = \log \hat{g}(k) + O(\log k).$$

We then show, unconditionally, that conjectures (1), (3), (4), and (5) above are true for $\hat{g}(k)$. (Note that it seems possible that conjecture (2) is true for $g(k)$ but false for $\hat{g}(k)$.) Specifically, we show that

$$0.525\ldots + o(1) \quad \leq \quad \frac{\log \hat{g}(k)}{k/\log k} \quad \leq \quad 1 + o(1),$$

which proves (3), (4), and (5), and we show that

$$\limsup_{k\to\infty} \frac{\hat{g}(k+1)}{\hat{g}(k)} = \infty.$$

Let $G(x,k)$ count the number of $n \leq x$ such that $p(\binom{n}{k}) > k$. We show unconditionally that, for $x > x_0(k)$,

$$G(x,k) = x/\hat{g}(k)(1 + o(1)).$$

This implies that $\hat{g}(k)$ should approximate $g(k)$ reasonably well.

With the assumption of our heuristic, we prove a running time for our algorithm of

$$g(k) \exp\left[-c\frac{k \log\log k}{(\log k)^2}\right]$$

for a constant $c > 0$. We also sketch an more general argument showing our algorithm running time is sublinear in $g(k)$, unconditionally.

Our paper is organized as follows. In §2 we present tables and graphs of our newly computed values of $g(k)$. In §3 we present Kummer's theorem

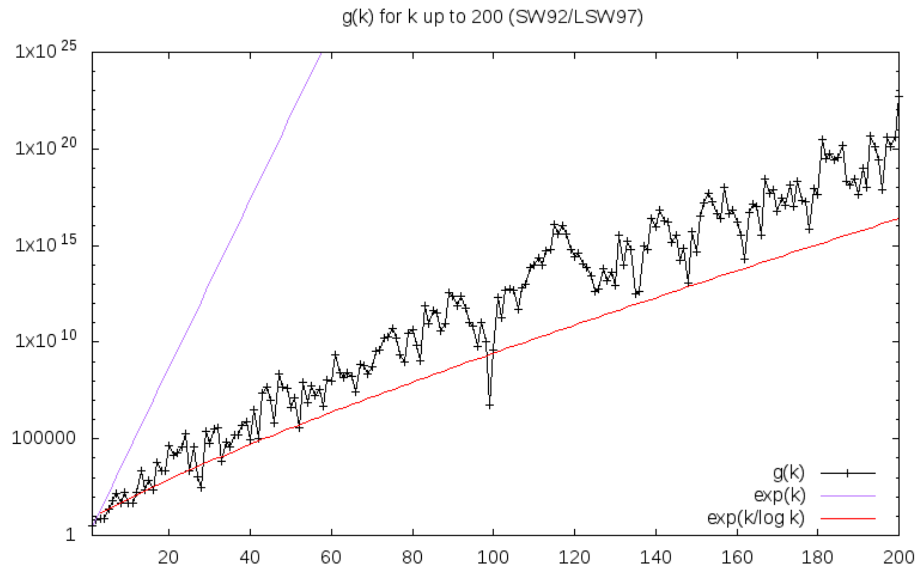

FIGURE 1. Logscale plot of $g(k)$ from [1, 15, 12].

and outline how our sieving algorithm works. In §4 we present our algorithm, including a description of the space-saving wheel sieve data structure, and an extended example. In §5 we discuss the knapsack subproblem and techniques for splitting prime rings when deciding the sieving modulus for the algorithm. In §6 we give our uniform distribution heuristic, provide some statistical evidence for its credibility, show that $g(k)$ is roughly $\hat{g}(k) = M_k/R_k$ with high probability, and we give an easy proof of our estimate for $G(x, k)$. In §7 we prove conjectures (1) and (3)-(5) for $\hat{g}(k)$ and bound the running time of our algorithm.

2. New Values of $g(k)$

Values of $g(k)$ we computed using a single processor core are listed in Table 1. Subsequent values of $g(k)$, computed using a small cluster with 192 cores, are in Table 2.

Walls of numbers are not to everyone's taste. In Figures 1 and 2 are logscale plots of $g(k)$ values.

3. Kummer's Theorem

We have the following.

k	$g(k)$	k	$g(k)$
200	520 87838 89271 01913 82732	225	12369 18109 50028 52853
201	235 54612 35023 12966 07453	226	1 33170 49136 16068 80243
202	371 93707 68876 94169 93998	227	25 43371 29078 24284 53367
203	36 66628 18040 77694 67119	228	4 42953 79137 83327 73614
204	178 22243 70804 75634 88989	229	74 31339 46454 40891 68359
205	119 23364 21369 70734 19215	230	1795 17836 21533 83405 06863
206	118 94994 19601 54916 70238	231	1535 32995 48871 64662 39991
207	4 14102 11738 06206 56623	232	111 43965 49911 64968 95483
208	128 63517 97975 53174 93464	233	20200 73550 49977 05129 39243
209	40 80254 70430 94462 56859	234	9141 02029 72226 64023 95374
210	3 81063 47274 59626 93595	235	3353 56843 86952 45592 15615
211	277 19087 51211 86811 93467	236	9004 68924 26010 08758 44863
212	254 11430 46501 91044 33623	237	128 10339 84890 50088 80623
213	941 02942 94951 13843 10999	238	797 67177 19809 53861 33999
214	5 42943 43587 62853 77239	239	18991 37758 35752 30838 29999
215	1 94050 01839 78664 31743	240	179 81118 13875 42559 25240
216	43 71951 29369 55065 01119	241	50194 81877 83204 04927 52119
217	17 60181 71551 23707 20217	242	1 05794 00205 01218 84737 54618
218	40 46933 47457 90358 45374	243	1675 50917 78080 00171 78623
219	4 60119 05176 06932 47999	244	7032 91964 49292 36074 24244
220	5 07302 74025 33237 33471	245	35619 19278 93870 30042 55997
221	11 24738 59029 35409 05471	246	57819 80943 94360 21432 63998
222	11720 13806 06713 22847	247	8791 54436 07103 73768 64247
223	29 34696 47028 07658 76223	248	1028 52788 08587 24965 75999
224	51633 58728 02682 87224	249	14 56775 25795 65720 74749

k	$g(k)$
250	276 46926 37489 69206 77374
251	3056 66797 58148 26766 11579
252	10505 00162 90998 95371 30494
253	3103 09358 30344 20590 94269
254	1166 62737 17826 44531 56094
255	1021 56121 82556 87267 01055
256	128 22340 15164 11349 38548
257	7712 29340 10480 52695 50483
258	58587 79034 00801 08562 54858
259	28954 56510 29429 20300 85999
260	9052 78792 60680 37520 93549
261	3752 17161 28291 37355 29917
262	370 44716 59526 93861 95399
263	52789 04430 06789 43127 33639
264	34644 90142 64935 39757 27919
265	29014 53224 87882 19896 83691
266	45629 29239 07110 01927 14698
267	8235 39060 08758 91988 86219
268	68 39739 60096 00722 01118
269	1 61558 09307 54284 34696 01199
270	17012 60056 85638 85052 85598
271	2 37245 88062 88508 66946 32223
272	57 61284 34192 78614 55093 37498

TABLE 1. Values of $g(k)$, for k up to 272, found with a single processor core.

k	$g(k)$
272	57 61284 34192 78614 55093 37498
273	11 93755 72096 07235 88168 84023
274	2 88454 13176 35913 24169 68574
275	17152 34131 63802 94572 85911
276	88030 17168 24411 10341 86038
277	453 93397 13957 10829 21927 70333
278	39 50539 72728 23202 00849 01718
279	2 66648 13175 24792 99862 36799
280	70874 78896 73459 57906 03609
281	123 66293 54022 28562 17374 11069
282	7 38297 70384 67082 65425 71838
283	37813 55429 48519 12235 53898 37243
284	6100 10364 48359 18395 19770 39199
285	78766 18312 18052 31134 42561 68735
286	747 51565 01679 14418 20981 52223
287	992 72191 61150 70855 53665 86719
288	5090 76951 48442 32227 73921 76099
289	4834 99245 99858 90424 83401 35289
290	580 10024 22391 77582 79250 69666
291	209 69391 29197 03178 94977 03719
292	174 18908 52958 77197 48733 28493
293	16639 09980 87532 46018 20569 16799
294	20223 01592 35223 93093 61644 12799
295	14858 57580 15296 66376 70445 68447
296	41418 90259 64755 93533 87671 33096
297	2412 51951 98121 56990 65688 86073
298	25619 63627 54642 94279 56273 35598
299	1832 43102 56640 25079 58634 93499
300	701 85519 63812 11947 39815 22430
301	113 92964 05228 07857 10715 23117
302	1742 88530 64455 07964 88047 54943
303	129 26741 47619 33558 63300 61679
304	13 26053 17393 60472 36038 80314
305	1 72946 53384 73935 85567 11859
306	3 51841 28928 12034 40626 05307
307	1779 34819 88869 76850 45198 63743
308	563 71964 39859 00813 40202 10998
309	98 07021 15457 23811 10525 81749
310	1 24437 81505 29347 17696 51070
311	1560 53896 22680 68278 05256 06711
312	1796 99278 95512 29968 42460 24124
313	3 00996 54176 68374 47827 87101 84313
314	1 87014 93014 72478 06122 67573 88094
315	1361 11485 02742 01184 89157 03743
316	1 03374 01931 39808 86145 47639 65949
317	68 85447 25707 42253 40215 24113 79199
318	6 86881 00807 03611 96229 81358 41598
319	11 86184 98065 18829 52817 06712 46719
320	1 40079 84256 27063 06819 06499 16746
321	2435 79072 21965 54229 62339 49121
322	15 25966 57699 53539 87155 01511 11623
323	1 69829 77104 46041 21145 63251 22499

TABLE 2. Values of $g(k)$ found with a cluster of 192 processor cores.

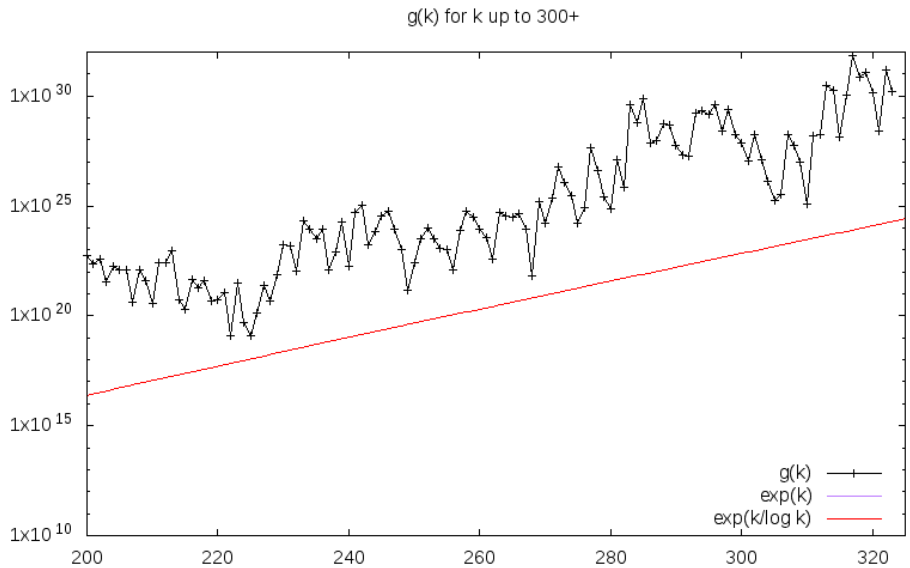

FIGURE 2. Logscale plot of our new $g(k)$ values.

Theorem 3.1. *Let $k < n$ be positive integers, and let p be a prime $\le k$. Let t be a positive integer with $t \ge \lfloor \log_p n \rfloor$. Write*

$$k = \sum_{i=0}^{t} a_i p^i \quad \textit{and} \quad n = \sum_{i=0}^{t} b_i p^i$$

as the base-p representations of k and n respectively. Then p does not divide $\binom{n}{k}$ if and only if $b_i \ge a_i$ for $i = 0, \dots, t$.

This primarily follows from Legendre's formula; a detailed proof is given in [16].

Example. Set $k = 10, n = 12, p = 5$. Writing in base 5, we have $k = 20_5$ and $n = 22_5$. This satisfies the theorem, so that $\binom{12}{10}$ is not divisible by 5, and indeed $\binom{12}{10} = 12 \cdot 11/2 = 66$. Changing p to 3, we have $k = 101_3$ and $n = 110_3$. We see that $1 = a_0 > b_0 = 0$, and so 3 divides $\binom{12}{10} = 66$.

Sieving Example. As was pointed out in [16], this allows us to sieve. We continue with the example $k = 10$.

For $p = 2$, we have $k = 1010_2$. We need all the $b_i \ge a_i$, so the choices are 1010_2, 1011_2, 1110_2, and 1111_2. That is, n must be $10, 11, 14$, or 15 modulo 16.

For $p = 3$, we have $k = 101_3$. This gives the 12 choices

$$101_3, 102_3, 111_3, 112_3, 121_3, 122_3, 201_3, 202_3, 211_3, 212_3, 221_3, 222_3$$

modulo $3^3 = 27$.

For $p = 5$, we have $k = 20_5$. This gives the 15 choices

$$20_5, 21_5, 22_5, 23_5, 24_5, 30_5, 31_5, 32_5, 33_5, 34_5, 40_5, 41_5, 42_5, 43_5, 44_5$$

modulo 25.

For $p = 7$, we have $k = 13_7$. This gives the 24 choices

$$\begin{aligned}&13_7, 14_7, 15_7, 16_7, 23_7, 24_7, 25_7, 26_7, 33_7, 34_7, 35_7, 36_7,\\ &43_7, 44_7, 45_7, 46_7, 53_7, 54_7, 55_7, 56_7, 63_7, 64_7, 65_7, 66_7\end{aligned}$$

modulo 49.

Applying the Chinese remainder theorem, this gives us $4 \cdot 12 \cdot 15 \cdot 24 = 17280$ admissible residues modulo $16 \cdot 27 \cdot 25 \cdot 49 = 529200$. Note that this equals M_{10} as defined in the Introduction. Define R_k to be the number of admissible residues modulo M_k, so that $R_{10} = 17280$. Then $g(k)$ is the smallest admissible residue $> k + 1$.

Continuing our example, it so happens that $g(10) = 46$. Checking, we have $46 \bmod 16 = 14$, $46 \bmod 27 = 19 = 201_3$, $46 \bmod 25 = 21 = 41_5$, and $46 \bmod 49 = 46 = 64_7$.

If the admissible residues are, more or less, evenly distributed modulo M_k, then we would expect $g(k) \approx \hat{g}(k) = M_k/R_k$. This is, in essence, our *uniform distribution heuristic*, which we discuss in §6. Note that $g(10) = 46$ and $\hat{g}(10) = M_{10}/R_{10} = 30.625$, so this is at best a rough approximation.

4. The Algorithm

The naive approach is to search through all the R_k admissible residues modulo M_k to find the smallest $> k + 1$. However, R_k is typically too large for this, making this algorithm practical only for very small k.

Instead, we enumerate residues that satisfy the requirements of Kummer's theorem modulo N, where N is a divisor of M_k that is larger than, but near to $g(k)$.

As we describe our algorithm, we continue our example with $k = 10$.

(1) Compute M_k, R_k, and estimate of $k\hat{g}(k) = k \cdot M_k/R_k$. For $k = 10$, this gives $M_{10} = 529200$, $R_{10} = 17280$, and an estimate of $10 \cdot 30.625 = 306.25$.

(2) Choose a divisor N of M_k just above our estimate. For $k = 10$, we choose $N = 16 \cdot 3 \cdot 7 = 336$.

N is chosen to have a good filtering rate to minimize the number of residues. Details of how to do this are discussed in §5.

(3) Build a *ring* data structure for each prime power dividing N. Basically, this is the list of admissible residues as defined by Kummer's theorem.

For $k = 10$ and $N = 336$, we have the following rings:

$10, 11, 14,$ or 15 modulo 16

1 or 2 modulo 3

$3, 4, 5,$ or 6 modulo 7

(4) Construct a *wheel* data structure to generate the residues modulo N. This algorithm is described in [17]. Below we show the jump tables computed for $k = 10$ and $N = 336$.

Ring 16:

residue	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
admissible	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1
jump	+10	+9	+8	+7	+6	+5	+4	+3	+2	+1	+1	+3	+2	+1	+1	+11

Ring 3:

residue	0	1	2
admissible	0	1	1
jump	+16	+16	+32

Ring 7:

residue	0	1	2	3	4	5	6
admissible	0	0	0	1	1	1	1
jump	+48	+96	+144	+192	+48	+48	+48

Each jump entry is the minimum amount to add that both preserves the residue class modulo earlier rings, and also jumps to an admissible residue for the current ring.

For speed, it is best to put the ring with the most residues last, but for correctness, the order does not matter.

(5) Rings for the remaining prime powers are also created, but not a wheel (the jumps are not needed). We refer to these rings as *filters*. A residue passes the filter if, when reduced modulo the ring size, the corresponding admissible bit is set to one. The smallest residue generated from the wheel that also passes all the filters is $g(k)$.

So for $k = 10$ and $N = 336$, we would build filters for 27, 25, and 49 at this step. Any prime power ring that is part of the wheel, where that prime power fully divides M_k, is not needed as a filter. Or in other words, if a prime divides N but not M_k/N, its prime power is not needed as a filter. So in our example, we require no filter for 16.

(6) Now that our data structures are initialized, we generate each residue modulo N from the wheel to see if it passes the filters. As we go, we maintain the value of the minimum residue, so far, that passed all the filters. Once every residue from the wheel is generated, this minimum is $g(k)$.

Example Continued. To see how the wheel works, we start with $k + 2$, 12 in our example, the smallest starting point. 12 is not admissible modulo 16, so we appy the jump (+2) to get 14. We pass up to the next ring. $14 \bmod 3 = 2$ is admissible. We pass to the next ring. $14 \bmod 7 = 0$ is not admissible, so we jump (+48) to get 62. There are 4 total residues in the 7 ring, so we also generate $62+48 = 110$, $110+48 = 158$, and $158+48 = 206$. All residues produced by the 7 ring are filtered:

$62 \bmod 27 = 8 = 22_3$, fail
$110 \bmod 27 = 2$, fail
$158 \bmod 27 = 23 = 212_3$ pass, but $158 \bmod 25 = 8 = 13_5$, fail

$206 \bmod 27 = 17 = 122_3$, pass, but $206 \bmod 25 = 6 = 11_5$, fail

We then backtrack to ring 3 at 14, and generate $14 + 32 = 46$. We pass to ring 7. The initial value in this ring, $46 \bmod 7 = 4$, is already admissible and is generated first. We also generate $46 + 48 = 94$, $94 + 192 = 286$, and $286 + 48 = 334$. These get filtered and 46 passes all filters. We record this value as a candidate for $g(10)$ and continue the computation to see if a smaller value exists. Since, $g(10) = 46$ no such value will be found. Note that nothing larger than N can be generated.

After 4 residues in the 7 ring, we drop down to the 3 ring, where we have already done 2 residues, so we drop back to the 16 ring. At the 16 ring, we generate the next residue $14 + 1 = 15$, which is passed up to the 3 ring.

This implies that, at each ring, we need to keep track of the next residue to generate, and how many have been generated so far so that we know when to back up to a previous ring.

And so it goes. The amortized cost is a constant number of arithmetic operations per residue generated by the outermost ring where they are filtered. If we apply the filters in decreasing order of filter rate, on average, a residue is only tested against a constant number of filters, and so again, the cost is a constant number of arithmetic operations per residue modulo N.

Finally, we mention that, by keeping track of the minimum residue that passes the filters, we don't have to generate any residues larger than this minimum. In our example, once we see 46 pass the filters, we don't even generate the rest of ring 7. This optimization can make a big difference in practice.

If we run the whole algorithm and fail to find a residue that passes the filters, this means $g(k) > N$. In this case, we simply multiply our estimate for $g(k)$ by k, choose a new, larger N, and try again.

Note that the problem of finding a solution below a given bound y to a system of pairwise coprime modular congruences is known to be NP-Complete. See [4, 13].

5. Prime Splitting and Knapsack

The purpose of this section is to look at how to choose N, a divisor of M_k that is just larger than our estimate for $g(k)$. We want to choose N so that the prime powers dividing N give a very low filter rate, thereby giving fewer residues to enumerate, which makes the algorithm faster.

Note that selecting prime power moduli based on filter rate alone is not optimal. The size of the modulus matters as well; a smaller modulus with a higher but still good filter rate can be preferable to a large modulus with a better filter rate.

We need some notation. Let $t_p := \lfloor \log_p k \rfloor + 1$ be the number of digits required to write k in base p, with the a_{ip} representing these digits, so that $k = \sum_{i=0}^{t_p - 1} a_{ip} p^i$. We have $t_p \geq 2$, and for most primes $t_p = 2$. Define T_p to be the maximum exponent of p so that $p^{T_p} \mid N$. This implies $0 \le T_p \le t_p$, and $N = \prod_{p<k} p^{T_p}$. Note that if k happens to be prime, it will have a terrible filtering rate, and so we never use it in N.

Let $r_{ip} := p - a_{ip}$, and let $R_{xp} := \prod_{i \le x} r_{ip}$. Then the number of acceptible residues modulo p^{T_p} is $R_{T_p p}$. The running time of the algorithm is proportional to the number of residues modulo N, which, by the Chinese remainder theorem, is

$$\prod_{p<k} R_{T_p p} = \prod_{p<k} p^{T_p} \frac{R_{T_p p}}{p^{T_p}} = N \cdot \prod_{p<k} \frac{R_{T_p p}}{p^{T_p}}.$$

We want to minimize the product of the filtering rates for primes included in N, which is equivalent to maximizing the reciprocal, which we write this way:

$$\prod_{p<k} \frac{p^{T_p}}{R_{T_p p}} = \exp \sum_{p<k} \log \frac{p^{T_p}}{R_{T_p p}}.$$

This allows us to set up a *knapsack problem* for choosing prime powers to include in N by setting the overall capacity of the knapsack to $\log N$, and the size and value of prime powers are set as follows:

$$\begin{aligned} \text{size}(p^T) &:= \log p^T = T \log p \\ \text{value}(p^T) &:= \log(\text{modulus}/\#\text{ residues}) = \log(p^T/R_T) = T \log p - \log R_T \end{aligned}$$

The question, then, is how to set T for each prime p to give a good selection of items to include in the knapsack. Also, we must insure that the same prime p is not chosen more than once, with different T values, for inclusion in the knapsack.

Asymptotically, we show in §7 that the expected size of $\log N$ is roughly $k/\log k$, so that only roughly $k/(\log k)^2$ primes are needed in N, allowing an average filter rate of about $1/\log k$ for each prime, and that T_p can be set to 1 for the primes included in N.

In practice, we can often get better results by including prime powers. So our approach is, for each prime $p < k$, to compute an optimal value for T based on filter rate, and then use a greedy algorithm to fill our knapsack. We call computing this value for T *splitting* the prime power, and label this split point s_p. We then allow for up to three possible choices for each prime p: set $T = 0$ (that is, omit p from N entirely), use $T = s_p$ (use the optimal split point), or use $T = t_p$, the maximum (note that $s_p = t_p$ is possible).

Next, we show how to compute the optimal split point s_p for each prime power, and then we give an example of its use in constructing N.

5.1. **Optimal Splitting.** Maximizing the value-to-size ratio, we get

$$\begin{aligned} \frac{\text{value}}{\text{size}} &= \frac{T \log p - \log R_{Tp}}{T \log p} \\ &= 1 - \frac{\log R_{Tp}}{T \log p}. \end{aligned}$$

So, in time linear in t_p, we can try all possible T values and quickly find the optimum, s_p. Also, since 1 and $\log p$ don't change, it suffices to compute $(1/T)\log R_{Tp}$ for each T to find the optimum.

Example. Continuing our example from above with $k = 10$, let us first look at $p = 2$. We have $k = 1010_2$. We compute $r_{12} = 2$, $r_{22} = 1$, $r_{32} = 2$, and $r_{42} = 1$. This gives $R_{12} = 2$, $R_{22} = 2$, $R_{32} = 4$, and $R_{42} = 4$. We get value-to-size ratios of 0, 1/2, 1/3, and 1/2. This implies $s_2 = 2$ or 4. In practice, we normally use the largest value for s_p when several values give the same ratio, since it implies a better filter rate.

For $p = 3$, we have $k = 101_3$. We have $r_{13} = 2$, $r_{23} = 3$, and $r_{33} = 2$. This gives $R_{13} = 2$, $R_{23} = 6$, and $R_{33} = 12$. The successive $(1/T)\log R$ values are $\log 2$, $(1/2)\log 6$, and $(1/3)\log 12$. Of these, $\log 2$ is the smallest, giving $s_3 = 1$.

In a similar fashion, we obtain $s_5 = 2$ and $s_7 = 1$.

We then construct the following table (using the natural logarithm):

p	T	*value*	*size*	*ratio*
2	4	$\log(2^4/4)$	$\log(2^4)$	0.5
3	1	$\log(3/2)$	$\log 3$	0.4009...
3	3	$\log(3^3/12)$	$\log(3^3)$	0.246...
5	2	$\log(5^2/20)$	$\log(5^2)$	0.069...
7	1	$\log(7/4)$	$\log 7$	0.287...
7	2	$\log(7^2/24)$	$\log(7^2)$	0.183...

Back in §4 we saw that we wanted N near 306 for $k = 10$. Using a greedy algorithm to choose the items to include in our knapsack of size $\log 306$, we first choose $2^4 = 16$, leaving $306/16 \approx 20$ "room" in our knapsack. We then choose 3 as the next-best item, leaving about $20/3 \approx 7$ room. The next best item is 7, filling all remaining room, and giving $N = 2^4 \cdot 3 \cdot 7$.

Remarks.

- The general knapsack problem is NP-complete, which means we currently do not have reasonably fast algorithms for this problem. Our approach to prime splitting and using a greedy algorithm to choose prime powers to include is heuristic. For more on the knapsack problem and the theory of NP-completeness, see [4, 9].
- Our problem of computing N is a bit different from the standard knapsack problem in that the size of our knapsack, $\log N$, is flexible, and we have items that are linked – if we choose 3^3 as an item, then we cannot choose 3^1, for example.
- We have a second method for splitting primes, which we call *contextual optimization* that we have not bothered to impement. The basic idea is to iterate over knapsack solutions, starting with a first solution based on the optimal splitting method described above.

  From that initial solution, we learn the overall quality of the solution, the global value-to-size ratio, and then when we resplit the primes we assume that a scaled version of this "background solution" will "fill in" for missing primes in our current prime power under examination. This can result in a different splitting point and potentially a better overall solution. This process is repeated until the knapsack solution stops improving.

We may or may not choose to explore this approach as we deal with larger and larger knapsack problems as k increases.

6. Uniform Distribution Heuristic

Let us recall some definitions and terminology.

We have $M_k := \prod_{p \le k} p^{\lfloor \log_p k \rfloor + 1}$. When writing k in base p, for a prime $p \le k$, we denote a_{ip} as the ith digit of k in base p, or $a_{ip} := \lfloor k/p^i \rfloor \bmod p$.

We say $r < M_k$ is an *admissible residue* if, for every prime $p \le k$, the digits of r, in base p, all exceed those of k in base p (satisfying the conditions of Kummer's lemma) so that p does not divide $\binom{r}{k}$ for every $p \le k$.

Let R_k be the total number of admissible residues $< M_k$. Then we have

$$R_k = \prod_{p \le k} \prod_{i=0}^{\lfloor \log_p k \rfloor} (p - a_{ip})$$

by the Chinese remainder theorem. $g(k)$, then, is the smallest r counted by R_k that is also larger than $k+1$.

Our estimate for $g(k)$, $\hat{g}(k)$, is defined as M_k/R_k.

6.1. The Uniform Distribution Heuristic (UDH).

We believe that the admissible residues behave as if they are chosen at random from a uniform distribution over the interval $[1, M_k - 1]$. This is our heuristic. It is not entirely dissimilar to the heuristic that integers $\le x$ are prime with probability $1/\log x$, and our intention is that these two models be treated similarly, in that we know they are not, stricty speaking, true, yet seem to have good predictive behavior under the right circumstances.

6.2. Evidence Supporting the UDH.

With the help of Rasitha Jayasekare, a statistician at Butler University, we ran statistical tests on the residues for $5 \le k \le 15$. The following table summarizes our findings.

k	R_k	Anderson-Darling	Kolmogorov-Smirnov
5	80	0.9885	1
6	96	0.9129	0.99
7	1008	1	0.978
8	2304	1	0.901
9	8640	1	0.945
10	17280	0.9989	1
11	285120	–	1
12	518400	–	0.994
13	8087040	–	1
14	9676800	–	1
15	16632000	–	0.998

We mapped the residues into the interval $(0, 1)$ by dividing them by M_k before running each test.

Here the Anderson-Darling column gives the p-value, a probability value between 0 and 1, that the given data came from a uniform distribution. The test fails at $k = 11$ and higher because the smaller residues were too close to zero, and the test takes the logarithm of the data items.

The Kolmogorov-Smirnov column reports p-values as well, and seems to tolerate very small values much better than the Anderson-Darling test.

For an introduction to statistical tests in the context of pseudorandom number generation, see [10, §3.3], where the Kolmogorov-Smirnov test is discussed in some detail.

6.3. **Estimating $g(k)$ with $\hat{g}(k)$.** Our approach is to first show that $g(k)$ is, with high probability, close to $\hat{g}(k) = M_k/R_k$. In the next section, we derive an estimate for M_k/R_k.

At this point, we will ignore admissible residues that are $\leq k+1$. Adjusting the derivation to include these means using, for example, $M_k - k$ and $R_k - k$ as appropriate below, but asymptotically this does not affect the rough estimates we obtain.

Theorem 6.1. *The UDH implies that, with probability* $1 - o(1)$*, we have*

$$\hat{g}(k)/k \leq g(k) \leq k\hat{g}(k).$$

Proof. We have

$$\begin{aligned} Pr(g(k) \leq x) &= 1 - Pr(\text{ all residues are } > x) \\ &= 1 - \left(\frac{M_k - x}{M_k}\right)^{R_k} \\ &= 1 - \left(1 - \frac{x}{M_k}\right)^{R_k} \end{aligned}$$

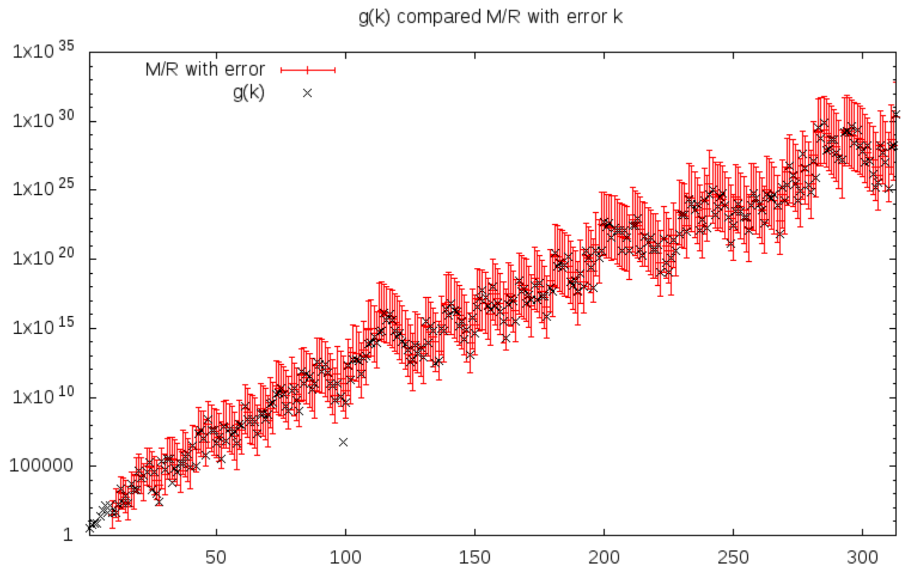

FIGURE 3. Comparing $g(k)$ with $\hat{g}(k)$

For an upper bound, set $x = (kM_k)/R_k$, to obtain

$$Pr(g(k) \leq (kM_k)/R_k) = 1 - \left(1 - \frac{k}{R_k}\right)^{R_k} \sim 1 - e^{-k} = 1 - o(1)$$

for large R_k (and R_k does get quite large).

Here we used the well-known fact that

$$\lim_{n\to\infty} \left(1 + \frac{x}{n}\right)^n = e^x.$$

For a lower bound, set $x = M_k/(kR_k)$ to obtain

$$Pr(g(k) \leq M_k/(kR_k)) = 1 - \left(1 - \frac{1}{kR_k}\right)^{R_k} \sim 1 - e^{-1/k} = o(1).$$

This completes the proof. $\square$

So we have that, with high probability,

$$\log g(k) = \log \hat{g}(k) + O(\log k)$$

if we assume the uniform distribution heuristic.

In Figure 3, we have empirical data comparing actual values of $g(k)$ (the black x's) compared to $\hat{g}(k)$ plotted as intervals from $\hat{g}(k)/k$ up to $k\hat{g}(k)$ as red error bars. The plot uses a logarithmic scale.

Figure 4 zooms in on the range $200 \leq k < 250$ for better visibility.

With the exception of $g(99)$, the data suggest that $\hat{g}(k)$ is a good estimator for $g(k)$.

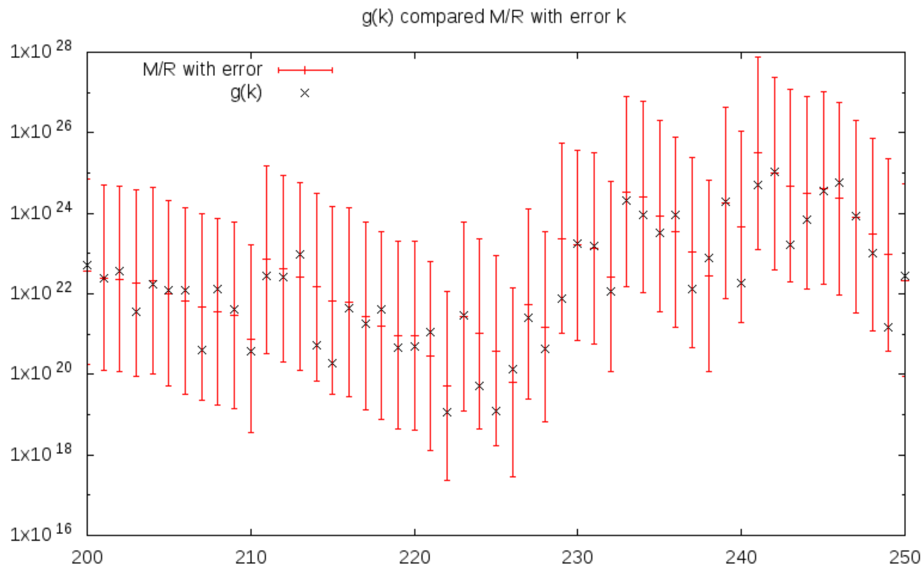

Figure 4. Comparing $g(k)$ with $\hat{g}(k)$ for $200 \leq k < 250$

Recall that $G(x, k)$ counts the integers $n \leq x$ such that $p(\binom{n}{k}) > k$. We conclude this section with the following.

Theorem 6.2. *If x is sufficiently large, then $G(x, k) = (x/\hat{g}(k))(1 + o(1))$.*

Proof. Write $x = q \cdot M_k + r$ using the division algorithm, with integers $q, r > 0$ and $r < M_k$. A contiguous interval of length M_k will have exactly R_k admissible residues, so $G(qM_k, k) = qR_k$. The remaining interval of length r has at most R_k residues, so $G(x, k) = G(qM_k, k) + O(R_k) = qR_k + O(R_k)$ but $q = \lfloor x/M_k \rfloor$, so

$$G(x, k) = \lfloor x/M_k \rfloor R_k + O(R_k) = (x/\hat{g}(k))(1 + o(1)).$$

□

7. Analysis

We start with a proof of Conjecture (1) from [1], but applied to $\hat{g}(k)$ instead of $g(k)$.

Theorem 7.1. *We have*

$$\limsup_{k\to\infty} \frac{\hat{g}(k+1)}{\hat{g}(k)} = \infty.$$

This proof uses some of the ideas from Section 3 in [12].

Proof. We will prove a lower bound proportional to $\log k$ in the case when $k + 1$ is an odd prime. Since there are infinitely many primes, this will be sufficient to prove the theorem.

Note that $\hat{g}(k+1)/\hat{g}(k) = (M_{k+1}/M_k)(R_k/R_{k+1})$.

First, we look at M_{k+1}/M_k. Recall that

$$M_k = \prod_{p \le k} p^{\lfloor \log_p k \rfloor + 1} \qquad \text{and} \qquad M_{k+1} = \prod_{p \le k+1} p^{\lfloor \log_p (k+1) \rfloor + 1}.$$

We can write

$$\begin{aligned} M_{k+1} &= \prod_{p \le k+1} p^{\lfloor \log_p (k+1) \rfloor + 1} \\ &= (k+1)^2 \cdot \prod_{p \le k} p^{\lfloor \log_p (k+1) \rfloor + 1} \\ &= (k+1)^2 \cdot M_k. \end{aligned}$$

Here we use the fact that for every prime $p \le k$, $\lfloor \log_p (k+1) \rfloor = \lfloor \log_p k \rfloor$ when $k+1$ is prime.

Next we look at R_k/R_{k+1}. Using the same notation for a_{ip} as above, and noting that the prime $k+1$ will contribute $k(k+1)$ residues, by Kummer's theorem, we have

$$\begin{aligned} \frac{R_k}{R_{k+1}} &= \frac{\prod_{p \le k} \prod_{i=0}^{\lfloor \log_p k \rfloor} (p - a_{ip})}{k(k+1) \cdot \prod_{p \le k} (p - (a_{0p} + 1)) \prod_{i=1}^{\lfloor \log_p (k+1) \rfloor} (p - a_{ip})} \\ &= \frac{1}{k(k+1)} \prod_{p \le k} \frac{p - a_{0p}}{p - (a_{0p} + 1)}. \end{aligned}$$

Again we note that $\lfloor \log_p (k+1) \rfloor = \lfloor \log_p k \rfloor$, and observe that the representation for $k+1$ in base p is the same as for k, with the exception of the least significant digit, a_{0p}, which is one larger, for all primes $p \le k$. This is only because $k+1$ is prime; $k+1 \bmod p$ cannot be zero unless $p = k+1$.

We then bound

$$\frac{p - a_{0p}}{p - (a_{0p} + 1)} \ge \frac{p}{p-1}$$

to obtain that

$$\frac{R_k}{R_{k+1}} \ge \frac{1}{k(k+1)} e^{\gamma} \log k (1 + o(1))$$

using Mertens's theorem. We deduce that

$$\frac{M_{k+1}/R_{k+1}}{M_k/R_k} \gg \frac{(k+1)^2}{k(k+1)} \log k \ge \log k$$

to complete the proof. □

To prove Conjectures (3), (4), and (5) from [1] for $\hat{g}(k)$, we prove the following.

Theorem 7.2.

$$0.525821\ldots + o(1) \le \frac{\hat{g}(k)}{k/\log k} \le 1 + o(1).$$

Applying the definitions for M_k and R_k above, we have

$$\begin{aligned}
\hat{g}(k) = \frac{M_k}{R_k} &= \frac{\prod_{p\le k} p^{\lfloor \log_p k\rfloor+1}}{\prod_{p\le k}\prod_{i=0}^{\lfloor \log_p k\rfloor}(p-a_{ip})} \\
&= \prod_{p\le k}\prod_{i=0}^{\lfloor \log_p k\rfloor}\frac{p}{p-a_{ip}} \\
&= \prod_{p\le\sqrt{k}}\prod_{i=0}^{\lfloor \log_p k\rfloor}\frac{p}{p-a_{ip}} \cdot \prod_{\sqrt{k}<p\le k}\prod_{i=0}^{\lfloor \log_p k\rfloor}\frac{p}{p-a_{ip}} \\
&= \prod_{p\le\sqrt{k}}\prod_{i=0}^{\lfloor \log_p k\rfloor}\frac{p}{p-a_{ip}} \cdot \prod_{\sqrt{k}<p\le k}\frac{p}{p-a_{1p}}\frac{p}{p-a_{0p}}.
\end{aligned}$$

Here we observed that $\lfloor \log_p k\rfloor + 1 = 2$ when $p > \sqrt{k}$.

We will show that the product on the factor involving a_{0p} is exponential in $k/\log k$, and is therefore significant; and the other two factors, the product on primes up to $\sqrt{k}$, and the factor with a_{1p}, are both only exponential in $\sqrt{k}$.

We bound the first product, on $p \le \sqrt{k}$, with the following lemma.

Lemma 7.3.

$$\prod_{p\le\sqrt{k}}\prod_{i=0}^{\lfloor \log_p k\rfloor}\frac{p}{p-a_{ip}} \quad \ll \quad e^{3\sqrt{k}(1+o(1))}.$$

This bound is not as tight as possible, but more than sufficient for our purposes.

Proof. We note that $a_{ip} \le p-1$, giving

$$\begin{aligned}
\prod_{p\le\sqrt{k}}\prod_{i=0}^{\lfloor \log_p k\rfloor}\frac{p}{p-a_{ip}} &\le \prod_{p\le\sqrt{k}}\prod_{i=0}^{\lfloor \log_p k\rfloor} p \quad = \quad \prod_{p\le\sqrt{k}} p^{\lfloor \log_p k\rfloor+1} \\
&\le \prod_{p\le\sqrt{k}} p^{3\lfloor \log_p \sqrt{k}\rfloor}.
\end{aligned}$$

From [7, Ch. 22] we have the bound

$$\sum_{p\le x}\lfloor \log_p x\rfloor \log p = x(1+o(1)). \tag{7.1}$$

Exponentiating and substituting $\sqrt{k}$ for x gives the desired result. ∎

Next, we show that the product involving a_{1p} is small.

Lemma 7.4.

$$\prod_{\sqrt{k}<p\leq k} \frac{p}{p-a_{1p}} \quad \ll \quad 2^{\sqrt{k}}.$$

This lemma is also not as tight as it might be; in particular, the 2 here can likely be replaced with $\sqrt{2}$. In any case, though, it seems clear from the proof that this is exponential in $\sqrt{k}$.

Proof. Observe that for any prime p with $\sqrt{k} < p \leq k$, if $a_{1p} = a$, then $k/(a+1) < p \leq k/a$. We have

$$\begin{aligned}
\prod_{\sqrt{k}<p\leq k} \frac{p}{p-a_{1p}} &= \prod_{a=1}^{\lfloor\sqrt{k}\rfloor} \prod_{k/(a+1)<p\leq k/a} \frac{p}{p-a} = \prod_{a=1}^{\lfloor\sqrt{k}\rfloor} \prod_{k/(a+1)<p\leq k/a} \left(1-\frac{a}{p}\right)^{-1} \\
&= \prod_{a=1}^{\lfloor\sqrt{k}\rfloor} \frac{\prod_{a<p\leq k/a}\left(1-\frac{a}{p}\right)^{-1}}{\prod_{a<p\leq k/(a+1)}\left(1-\frac{a}{p}\right)^{-1}} \\
&= \prod_{a=1}^{\lfloor\sqrt{k}\rfloor} \frac{(c(a)\log(k/a))^a(1+o(1))}{(c(a)\log(k/(a+1)))^a(1+o(1))} \\
&= (1+o(1))\frac{\log k}{\log(k/2)} \cdot \left(\frac{\log(k/2)}{\log(k/3)}\right)^2 \cdot \left(\frac{\log(k/3)}{\log(k/4)}\right)^3 \cdots \left(\frac{\log(\frac{k}{\lfloor\sqrt{k}\rfloor})}{\log(\frac{k}{\lfloor\sqrt{k}\rfloor+1})}\right)^{\lfloor\sqrt{k}\rfloor} \\
&= (1+o(1))\frac{\log k}{\log\sqrt{k}} \cdot \frac{\log(k/2)}{\log\sqrt{k}} \cdot \frac{\log(k/3)}{\log\sqrt{k}} \cdots \frac{\log(k/\lfloor\sqrt{k}\rfloor)}{\log\sqrt{k}} \\
&\ll 2^{\sqrt{k}}.
\end{aligned}$$

This used the following variant of Mertens's theorem, which holds for $b > 0$, where $c(b)$ is a constant that depends only on b:

$$\prod_{b<p\leq x}\left(1-\frac{b}{p}\right) = \left(\frac{c(b)}{\log x}\right)^b (1+o(1)). \tag{7.2}$$

This is readily proved following the arguments in Hardy and Wright [7, §22.7]. ☐

We now have

$$\log \hat{g}(k) = \log\left(\prod_{\sqrt{k}<p<k} \frac{p}{p-a_{0p}}\right) + O(\sqrt{k}).$$

The following lemma, then, wraps up the proof of our theorem.

Lemma 7.5.

$$0.525821\ldots \cdot \frac{k}{\log k}(1+o(1)) \le \log\left(\prod_{\sqrt{k}<p\le k} \frac{p}{p-a_{0p}}\right) \le \frac{k}{\log k}(1+o(1)).$$

Proof. Fix $a_{1p}=a$. Then $k/(a+1)<p\le k/a$, and $a_{0p}=k \bmod p = k-ap$ and $p-a_{0p}=p-(k-ap)=(a+1)p-k$. We have

$$\begin{aligned}\prod_{k/(a+1)<p\le k/a} \frac{p}{p-a_{0p}} &= \prod_{k/(a+1)<p\le k/a} \frac{p}{(a+1)p-k} \\ &= \exp \sum_{k/(a+1)<p\le k/a} \log(p)-\log((a+1)p-k)\end{aligned}$$

The first term, then, is $k/(a(a+1))+o(k/\log k)$, using

$$\sum_{p<x} \log p = x + o(x/\log x). \tag{7.3}$$

Rewriting the second sum as an integral, using the prime number theorem, we get

$$\begin{aligned}&- \sum_{k/(a+1)<p\le k/a} \log((a+1)p-k) \\ =& -\int_{k/(a+1)}^{k/a} \frac{\log((a+1)t-k)}{\log t} dt + o(k/\log k) \\ =& -\frac{1}{\log(k/(a+\alpha))} \int_{k/(a+1)}^{k/a} \log((a+1)t-k)dt + o(k/\log k)\end{aligned}$$

Here α is between 0 and 1, determined implicitly by the mean value theorem. The precise value of α may depend on both k and a. We use either $\alpha=0$ or $\alpha=1$, depending on whether we want an upper or lower bound, respectively.

Using substitution, we can readily show that

$$\int_{k/(a+1)}^{k/a} \log((a+1)t-k)dt = \frac{k(\log(k/a)-1)}{a(a+1)}.$$

We have, then,

$$\begin{aligned}\log\left(\prod_{\sqrt{k}<p<k} \frac{p}{p-a_{0p}}\right) &+ o(k/\log k) \\ &= \sum_{a=1}^{\sqrt{k}} \left(\frac{k}{a(a+1)} - \frac{k(\log(k/a)-1)}{a(a+1)\log(k/(a+\alpha))}\right) \\ &= \frac{k}{\log k}\cdot \sum_{a=1}^{\sqrt{k}} \frac{1-\log\left(1+\frac{\alpha}{a}\right)}{a(a+1)} \cdot \left(1+O\left(\frac{\log a}{\log k}\right)\right).\end{aligned}$$

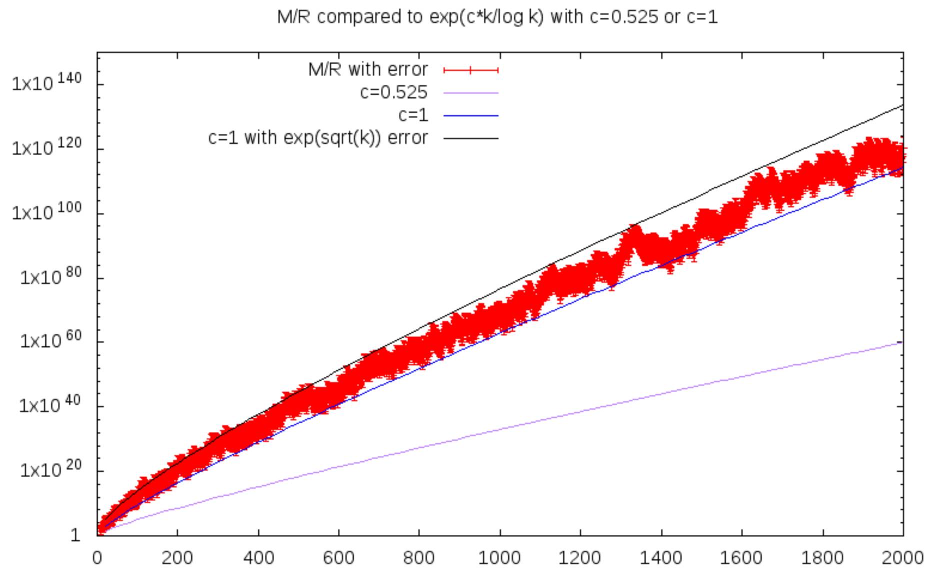

Figure 5.

The last step requires a bit of algebra, and the observation that $1/(u-v) = 1/u + v/(u(u-v))$. Also note that the error term is truly error, as can be seen by splitting the sum at, say, $(\log k)^2$.

To obtain the upper bound, set $\alpha = 0$, and note that $\sum 1/(a(a+1))$ converges to 1. To obtain the lower bound, set $\alpha = 1$, and note that $\sum(1 - \log(1+1/a))/(a(a+1))$ converges to a constant near $0.525821\ldots$. □

We do not know if the limit

$$\lim_{k\to\infty} \frac{\log \hat{g}(k)}{k/\log k}$$

exists. See Figure 5, which plots $\hat{g}(k)$ for $k \leq 2000$, and compares it to graphs of the functions $\exp[ck/\log k]$ for $c = 0.525\ldots$ and $c = 1$, and to $\exp[k/\log k + \sqrt{k}]$ to account for the error terms above that are exponential in $\sqrt{k}$.

Algorithm Running Time. We conclude with a bound on the running time of our algorithm.

Theorem 7.6. *If the UDH is true, then with probability* $1 - o(1)$*, our algorithm has a running time bounded by*

$$g(k) \cdot \exp\left[\frac{-ck \log\log k}{(\log k)^2}(1 + o(1))\right]$$

where $c > 0$ *is constant.*

Proof. Without loss of generality, we assume that $g(k) \le N < k \cdot g(k)$, as we can guess a smaller N, run the algorithm, and if it fails to find $g(k)$, include another prime p with $k/2 < p < k$ in N, and repeat. Since N at least doubles each time we do this, the cost of running the algorithm on all $N < g(k)$, and failing, is bounded by a factor of $\log g(k)$ times the cost of the final run with a value of $N > g(k)$ that succeeds. We absorb this multiplicative factor of $\log g(k)$ in the $o(1)$ error term in the exponent of the running time bound above as $\log g(k) = \Theta(k/\log k)$ with high probability. In particular, this gives us $\log N = (1 + o(1)) \log g(k)$ with high probability.

For the purposes of this proof, we choose N to be a product of some primes between $k/2$ and k. This is conservative, as the choice of primes or prime powers for inclusion in N, using the methods discussed earlier, will result in a faster algorithm in practice. So we have

$$\prod_{p|N} p = N \approx g(k)$$

and thus

$$\sum_{p|N} \log p = \log N \sim \log g(k) \ll k/\log k.$$

Since $\sum_{k/2<p\le k} \log p = (k/2)(1 + o(1))$, we have more primes in this range than we need for N by a factor of roughly $(1/2) \log k$. Thus, we can choose the best $k/(\log k)^2$ primes (roughly) below k of the $k/\log k$ that are available. As a result, we expect to get a filtering factor of $1/\log k$ for the primes we choose. Indeed, if we choose all primes p with $k/2 < p < k/2 + c_1k/\log k$, with $c_1 > 0$ an appropriate constant we fix later, this is the case.

Let's check that this gives us a good value for N. We have

$$\begin{aligned}
\log N &= \sum_{k/2<p<k/2+c_1k/\log k} \log p \\
&= \frac{c_1k}{(\log k)^2} \log(k/2)(1 + o(1)) \\
&= \frac{c_1k}{\log k}(1 + o(1)),
\end{aligned}$$

which is larger than $\log g(k)$ with high probability if we choose $c_1 > 2$. (See [14, (2.29)].)

Now we address the filter rate, and hence the running time. For each such prime p,

$$k + \frac{2c_1k}{\log k} > 2p > k,$$

which implies

$$k - p > p - \frac{2c_1k}{\log k}$$

so that

$$\begin{aligned} a_{0p} &= k \bmod p &=& \quad k-p \\ &> p - \frac{2c_1 k}{\log k} &>& \quad p - \frac{4c_1 p}{\log k} \\ &= p\left(1 - \frac{4c_1}{\log k}\right). \end{aligned}$$

Our running time, then, is proportional to the number of acceptible residues modulo N, which is

$$\begin{aligned} \prod_{k/2<p<k/2+c_1k/\log k} (p - a_{0p}) &= \prod_p \left(p - p\left(1 - \frac{4c_1}{\log k}\right)\right) \\ &= \prod_p p \cdot \frac{4c_1}{\log k} \\ &= N \prod_p \frac{4c_1}{\log k} \\ &\le kg(k) \left(\frac{4c_1}{\log k}\right)^{c_1 k/(\log k)^2 (1+o(1))} \\ &= g(k) \exp\left[-c_1 \frac{k \log\log k}{(\log k)^2}(1+o(1))\right]. \end{aligned}$$

□

The UDH is stronger than what we need to prove a sublinear running time. The central issue is finding enough primes p with $k/2 < p \le k/2 + \Delta$ such that the product of these primes is roughly $g(k)$. If the number of primes in this interval is $\Delta/\log k$, then we can set $\Delta \approx \log g(k)$. Pushing this through our argument above, we obtain a running time of the form

$$g(k) \cdot \exp\left[\frac{-c\Delta}{\log k} \log\left(\frac{4\Delta}{k}\right)(1+o(1))\right]$$

where $c > 0$ is constant. Observe that plugging in $\log g(k) \approx k/\log k$ gives our theorem, but this form is valid so long as we can find enough primes. In fact, if $\log g(k) \gg k^\theta$, with $7/12 < \theta \le 1$, we can use a result due to Heath-Brown [8] on primes in short intervals to guarantee this is true.

If $g(k)$ is smaller than this, we would choose $\Delta = (\log g(k)/\log k)E(k)$, where $E(k)$ is the error term for the prime number theorem for $\pi(k)$, to give us the needed $\log g(k)/\log k$ primes above $k/2$. (If we assumed the Riemann Hypothesis, this would let us use a smaller $E(k)$ term.) Pushing this through, we obtain a weaker, but still sublinear, running time.

Acknowledgments. The first author was supported in part by the Butler Summer Institute, by the Honors program, and by the Mathematics Research Camp at Butler University. The second and third authors were supported in part by a grant from the Holcomb Awards Committee at Butler University.

Special thanks to Rasitha Jayasekare, our friendly neighborhood statistician, for helping us with uniform distribution statistical tests. Also thanks to Michael Filaseta for his help with references.

Finally, thanks to Frank Levinson, who generously supports Butler University's computing research infrastructure.

References

[1] E. F. Ecklund, Jr., P. Erdös, and J. L. Selfridge. A new function associated with the prime factors of $\binom{n}{k}$. *Math. Comp.*, 28:647–649, 1974.

[2] P. Erdős, C. B. Lacampagne, and J. L. Selfridge. Estimates of the least prime factor of a binomial coefficient. *Math. Comp.*, 61(203):215–224, 1993.

[3] Paul Erdős. Some problems in number theory. In A.O.L. Atkin and B.J. Birch, editors, *Computers in Number Theory*, pages 405–414. Academic Press, 1971.

[4] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[5] Andrew Granville and Olivier Ramaré. Explicit bounds on exponential sums and the scarcity of squarefree binomial coefficients. *Mathematika*, 43(1):73–107, 1996.

[6] Richard K. Guy. *Unsolved problems in number theory*. Problem Books in Mathematics. Springer-Verlag, New York, third edition, 2004.

[7] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, 5th edition, 1979.

[8] D.R. Heath-Brown. The number of primes in a short interval. *Journal für die reine und angewandte Mathematik*, 389:22–63, 1988.

[9] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer Berlin Heidelberg, 2013.

[10] D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley, Reading, Mass., 3rd edition, 1998.

[11] S. V. Konyagin. Estimates of the least prime factor of a binomial coefficient. *Mathematika*, 46(1):4155, 1999.

[12] Richard F. Lukes, Renate Scheidler, and Hugh C. Williams. Further tabulation of the Erdős-Selfridge function. *Math. Comp.*, 66(220):1709–1717, 1997.

[13] Kenneth L. Manders and Leonard Adleman. NP-complete decision problems for binary quadratics. *Journal of Computer and System Sciences*, 16(2):168 – 184, 1978.

[14] J. B. Rosser and L. Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois Journal of Mathematics*, 6:64–94, 1962.

[15] R. Scheidler and H. C. Williams. A public-key cryptosystem utilizing cyclotomic fields. Technical Report 15/92, University of Manitoba, Department of Computer Science, November 1992.

[16] Renate Scheidler and Hugh C. Williams. A method of tabulating the number-theoretic function $g(k)$. *Math. Comp.*, 59(199):251–257, 1992.

[17] Jonathan P. Sorenson. The pseudosquares prime sieve. In Florian Hess, Sebastian Pauli, and Michael Pohst, editors, *Proceedings of the 7th International Symposium on Algorithmic Number Theory (ANTS-VII)*, pages 193–207, Berlin, Germany, July 2006. Springer. LNCS 4076, ISBN 3-540-36075-1.

[18] Jonathan P. Sorenson. Sieving for pseudosquares and pseudocubes in parallel using doubly-focused enumeration and wheel datastructures. In Guillaume Hanrot, Francois Morain, and Emmanuel Thomé, editors, *Proceedings of the 9th International Symposium on Algorithmic Number Theory (ANTS-IX)*, pages 331–339, Nancy, France, July 2010. Springer. LNCS 6197, ISBN 978-3-642-14517-9.

[19] Jonathan P. Sorenson and Jonathan Webster. Strong pseudoprimes to twelve prime bases. *Math. Comp.*, 86(304):985–1003, 2017.

[20] Jonathan P. Sorenson and Jonathan Webster. Two algorithms to find primes in patterns. arXiv:1807.08777, 2018.

Butler University, Indianapolis, IN 46208, USA
E-mail address: bsorenso@butler.edu

Butler University, Indianapolis, IN 46208, USA
E-mail address: sorenson@butler.edu
URL: blue.butler.edu/∼jsorenso

Butler University, Indianapolis, IN 46208, USA
E-mail address: jewebste@butler.edu