Algorithmic Number Theory Symposium XVII, Groningen, July 6-10, 2026

NUMERICAL VERIFICATION OF THE COLLATZ CONJECTURE FOR BILLION DIGIT RANDOM NUMBERS

ANDREAS-STEPHAN ELSENHANS

Abstract. The Collatz conjecture, also known as the $3n+1$ problem, is one of the most popular open problems in number theory. This note reports on an algorithm for the verification of the Collatz conjecture that can handle numbers with a billion decimal digits on a standard PC.

Some statistics on the total stopping times observed is presented. Furthermore, the run time is compared with the run times reported in earlier investigations in this direction.

1. Introduction

The Collatz conjecture asserts that, for any positive integer n_0, the sequence defined by the recurrence relation

$$n_{k+1} = \begin{cases} \frac{n_k}{2} & \text{if } n_k \text{ is even,} \\ \frac{3n_k+1}{2} & \text{if } n_k \text{ is odd,} \end{cases}$$

will always eventually reach the value 1, regardless of the initial value n_0. Despite its simple formulation, the conjecture has resisted all attempts at a general proof, though it has been verified [2] for all integers up to 2^{71}.

This article considers the numerical verification from a different perspective. Instead of checking the conjecture for all numbers up to a given bound, the aim is to check extremely large numbers. Empirically, a check of the conjecture for an ℓ-bit number working directly with the definition takes $O(\ell^2)$ bit operations (cf. Remark 5.5). However, the algorithm presented in this note reduces this to $\widetilde{O}(\ell)$ bit operations (cf. Theorem 5.8). This improved asymptotic run time allows for a check of billion digit numbers on a standard PC. Testing large numbers is now limited by available memory, whereas earlier approaches were limited by CPU time.

A rigorous proof of the observed complexity bound would, imply the Collatz conjecture. Therefore, the given proof relays on a quantitative version of the Collatz conjecture.

Furthermore, some statistics on the number of iterations to reach 1 is presented. It indicates that this number is about $\log(n_0)/\log(2/\sqrt{3})$ and that the distribution is similar to a normal distribution.

The programs used and the data generated are available via:
`https://www.mathematik.uni-wuerzburg.de/en/computeralgebra/team/elsenhans-stephan-prof-dr/research-supplements/`

Key words and phrases. Numerical verification of the Collatz conjecture using fast arithmetic.

2. Related work

A lot of work has been done on the verification of the Collatz conjecture for all integers up to a given bound [1, 2, 6]. Here, the main idea is that one has to compute the iteration until a smaller number is reached. The number of steps necessary for this is called the *stopping time*, whereas the number of steps to reach 1 is called the *total stopping time*.

The check that one actually has a finite stopping time allows a large number of optimisations. First, all even numbers can be skipped. Furthermore, a number $n \equiv 1 \bmod 4$ will result in $(3n+1)/4$ after two iterations. Thus, only the residue class $n \equiv 3 \bmod 4$ needs a closer look. This concept of looking at a number after several iterations is called *compression*.

The above can indeed be extended. For example, only the residue classes 7, 15, 27, and 31 modulo 32 need to be checked. In other words, only these residue classes do not lead to a smaller number within the first five steps. In general, for all numbers n in a fixed residue class modulo 2^m, the first m steps of the Collatz sequence are given by

$$\frac{3^{e_1}n + z_1}{2}, \ldots, \frac{3^{e_m}n + z_m}{2^m},$$

where the polynomials $\frac{3^{e_j}X + z_j}{2^j}$ are called the *standard polynomials* [6, Sec. 2.3.2].

Our algorithm will use the same approach. I.e., it will encode a large number of iterations of the Collatz sequence into a standard polynomial. The polynomials can be handled efficiently by using suitable multiplication algorithms.

3. Standard Polynomials

3.1. **Definition.** The function

$$T\colon \mathbb{N} \to \mathbb{N}, n \mapsto \begin{cases} \frac{n}{2} & \text{if } n \text{ is even,} \\ \frac{3n+1}{2} & \text{if } n \text{ is odd,} \end{cases}$$

is called the *Collatz T function*. We call a direct evaluation of the T function an *elementary step* of the Collatz iteration.

3.2. **Fact.** *For every $N := 2^k$, there exist unique linear polynomials $S_{k,r} \in \mathbb{Q}[X]$ $(r = 0, \ldots, N-1)$ such that*

$$S_{k,r}(n) = T^{(k)}(n)$$

for all $n \equiv r \pmod N$. Here, $T^{(k)}$ denotes the k-th iteration of the Collatz T function.

Proof. [6, Sec. 2.3.2]. □

3.3. **Definition.** The polynomials $S_{k,r}$ are called *standard polynomials*.

3.4. **Fact.** *The standard polynomials satisfy the relation*

$$S_{k+l,r}(X) = S_{k, S_{l, r \bmod 2^l}(r) \bmod 2^k}(S_{l, r \bmod 2^l}(X)),$$

for all positive integers k, l and $r \in \{0, \ldots, 2^{k+l} - 1\}$.

Proof. This is the equality $T^{k+l} = T^k \circ T^l$, restricted to the residue class r modulo 2^{k+l}, encoded in standard polynomials. □

3.5. **Application.** For $k = 2$, the standard polynomials are

$$S_{2,0} = \frac{X}{4},\ S_{2,1} = \frac{3X+1}{4},\ S_{2,2} = \frac{3X+2}{4},\ S_{2,3} = \frac{9X+5}{4}\,.$$

One can read off these polynomials once more that only the residue class 3 modulo 4 contains integers larger than 4 that do not lead to a smaller value after two iterations. Furthermore, for $k = 5$, one has

$$\begin{aligned} S_{5,3} &= \frac{9X+5}{32},\ S_{5,7} = \frac{81X+73}{32},\ S_{5,11} = \frac{27X+23}{32},\\ S_{5,15} &= \frac{81X+65}{32},\ S_{5,19} = \frac{27X+31}{32},\ S_{5,23} = \frac{27X+19}{32},\\ S_{5,27} &= \frac{81X+85}{32},\ S_{5,31} = \frac{243X+211}{32}\,. \end{aligned}$$

One can read off these polynomials that only the residue classes $7, 15, 27$, and 31 modulo 32 do not lead to a smaller value within the first five iterations, as the residue classes 0, 1, and 2 modulo 4 are already eliminated by the previous list of standard polynomials.

One can extend this easily and determine the residue classes modulo 2^k that do not lead to a smaller value within the first k iterations. The numbers of such classes are listed in the table below:

Modulus	# remaining classes	rem. (%)	Modulus	# remaining classes	rem. (%)
2	1	50	2^2	1	25
2^3	2	25	2^4	3	18.75
2^5	4	12.5	2^6	8	12.5
2^7	13	10.16	2^8	19	7.42
2^9	38	7.42	2^{10}	64	6.25
⋮	⋮	⋮	⋮	⋮	⋮
2^{25}	573162	1.708	2^{26}	1037374	1.546
2^{27}	1762293	1.313	2^{28}	3524586	1.313
2^{29}	6385637	1.189	2^{30}	12771274	1.189

The sequence `https://oeis.org/A076227` contains the number of remaining residue classes. A speed up by a factor of 14 is reported in [6] by the restriction to the remaining residue classes.

4. An algorithm for large numbers

4.1. **Introduction.** It is well known that, when working with large numbers, special multiplication algorithms [9] have to be used. Nowadays, these are available in various computer algebra systems and libraries [5]. As the potential of such methods has to be utilised here, the computer algebra system `magma` [3] was used in this investigation.

4.2. **Data structure and code.** As all standard polynomials are of the shape $\frac{aX+b}{2^c}$, the triple $\langle a, b, c\rangle$ was used to represent them. This leads to the following basic functions:

```
// Evaluate the polynomial p (given as a triple) at the integer x
// assuming that the result is an integer
function MyEval(p,x)
 return ShiftRight(x * p[1] + p[2],p[3]);
end function;

// Compute p(q(X)) for polynomials in triple representation
function MySubs(p,q)
 return <p[1]*q[1], p[1]*q[2] + ShiftLeft(p[2],q[3]), p[3]+q[3]>;
end function;

// Direct construction of the standard polynomial S_{l,r}
function PolyDirect(r,l)
 pol := <1,0,0>;
 for j := 1 to l do
  if IsEven(r) then
   r := r div 2;            pol[3] := pol[3] + 1;
  else
   r := (3*r+1) div 2;      pol := MySubs(<3,1,1>,pol);
  end if;
 end for;
 return pol;
end function;

// Store the first standard polynomials in a list for fast access
PolList := [[PolyDirect(i,l) : i in [0..2^l-1]] : l in [1..8]];
```

Using this and Fact 3.4, one can compute the standard polynomials for large moduli by a binary splitting method:

```
// Fast recursive construction of S_{l,r}
function PolyFast(r,l)
 if l le 8 then return PolList[l][(r mod 2^l) + 1]; end if;
 t1 := l div 2;    t2 := l - t1;
 p1 := PolyFast(ModByPowerOf2(r,t1), t1);
 p2 := PolyFast(ModByPowerOf2(MyEval(p1,ModByPowerOf2(r, l)),t2),t2);
 return MySubs(p2,p1);
end function;
```

Now, one can check the Collatz conjecture for large random numbers by jumping from n to its $k := \lfloor \log_2(n)/2 \rfloor$-th successor $T^k(n)$ in one *macro step*. In other words, the number of elementary steps in one macro step is linear in the bit length of the current number.

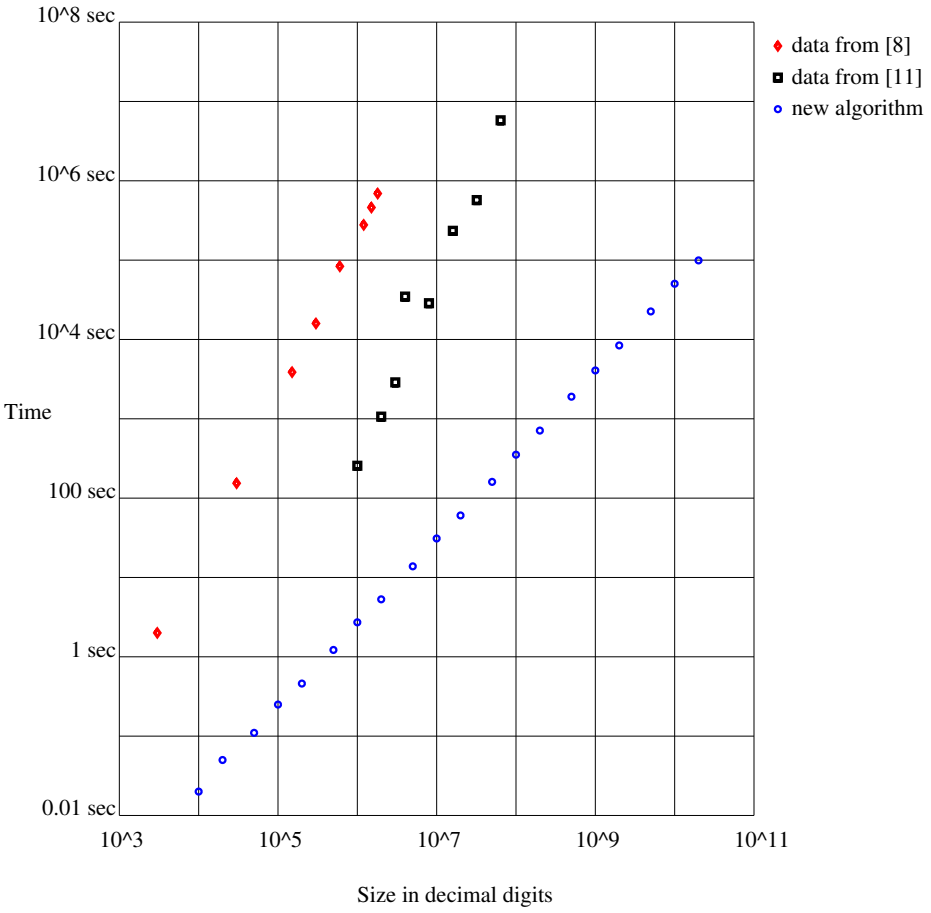
```
 len := 10^5;   n := Random(10^len);     cnt := 0;
 time repeat
  StepSize := Max(1,Ilog2(n) div 2);    cnt := cnt + StepSize;
  pol := PolyFast(n,StepSize);          n := MyEval(pol,n);
 until n eq 1;
 len, cnt;
```

4.3. **Results.** The code above was applied to large random numbers having up to several billion decimal digits. In all cases, the iteration ends up in 1. The run time for one number and a statistic on the number of iterations to reach 1 are listed in the table below. All computations are performed on one core of an Intel i7-12700 processor running at 4.7 GHz.

Number of decimal digits	Number of examples	Average Number of steps to reach 1	Standard Deviation	Time
10 000	10000	160 085	1531	0.02s
100 000	10000	1 600 728	4838	0.25s
1 000 000	10000	16 007 868	15177	2.72s
10 000 000	10000	160 077 568	48567	31s
100 000 000	4000	1 600 785 302	152901	353s
1 000 000 000	400	16 007 824 543	515453	1.13h
10 000 000 000	1	160 079 821 246	—	13.2h

The statistics above indicates that the average number of elementary steps to reach 1 is about $\gamma \cdot \log(n_0)$ and the standard deviation is about $\gamma^{3/2} \cdot \sqrt{\log(n_0)} \cdot \log(3)/2$, for $\gamma := 1/\log(2/\sqrt{3})$. Furthermore, the observed skewness is between -0.022 and 0.054 and the observed kurtosis is between 2.967 and 3.075. A comparison of the observed distribution and a normal distribution can be done by computing the L^∞-distance of the cumulative distribution functions. This is known as *Kolmogorov–Smirnov test* [10]. In our case, it does not detect a significant difference between the distribution observed and a normal distribution.

4.4. **Comparison with other implementations.** To assess the new algorithm, its run time is compared with the run times reported by other authors. The plot below presents the run time as a function of the input size on logarithmic scales for both axes, which visualises the growth more clearly.



One can read off the graph that the run time of previous algorithms is quadratic in the size of the input whereas the new method is almost linear. Note that the investigations in [1, 2, 6] do not consider large numbers and therefore the reported performance is outside the plot range. Furthermore, those investigations focus on the stopping time, not the total stopping time.

One might also ask how this algorithm performs when implemented in `C`. For this, all integers exceeding the size of a `long int` have to be handled with a library such as GMP [5]. An optimised implementation runs about 7.5 times faster for numbers with a million digits. However, for numbers with a billion digits, the speed up is only a factor of 3.5. Thus, the interpreter overhead is much smaller than one might expect.

4.5. **Optimal macro step size.** To determine the optimal macro step size experimentally, the test number $n_0 = 13^{10^6} - 1$ was chosen and the size of the macro steps was set to $\mathrm{Max}(1, \lfloor \log_2(n)/D \rfloor)$. The table below lists the run time to verify the Collatz conjecture for n_0 using the `magma` and the `C` programs, for various values of D:

D	10	6	5	4	3	2	4/3	1
`magma` time in sec.	3.11	3.08	3.03	3.01	3.01	3.04	3.05	3.06
`C` time in sec.	0.48	0.41	0.41	0.40	0.41	0.42	0.45	0.47

This indicates that the optimal value of D is between 2 and 5. Other test numbers lead to the same conclusion. Larger macro steps result in larger and more expensive standard polynomials, whereas smaller macro steps would result in too many expensive evaluations of the standard polynomials in the main loop.

Remark. Starting with the number 2^k, the Collatz iteration reaches 1 within k elementary steps. Any other $(k+1)$-bit number results in a longer Collatz sequence. This shows that $D = 1$ is the smallest possible value, as otherwise the last macro step may go beyond the first occurrence of 1 in the Collatz iteration.

4.6. **Memory usage.** The largest number checked by the author had $2 \cdot 10^{10}$ decimal digits (about 8GB in binary representation). The computation (using `magma`) took about 28 hours and used 70 GB of memory. Thus, the limitation of checking large numbers is now given by the available memory, no longer by the available CPU time.

5. Complexity

5.1. **Introduction.** The graph above suggests that the run time of the algorithm presented is $\widetilde{O}(\ell)$. Here, ℓ denotes the bit length of the input. A full proof of this would imply the Collatz conjecture. Therefore, we establish the complexity only for all numbers that satisfy a quantitative version of the Collatz conjecture. We formalise this quantitative version by introducing a set $G_c \subset \mathbb{N}$, for every $c > 1$.

5.2. **Definition.** We denote by $B(n)$ the bit length of an integer n. For any constant $c > 1$, we put

$$G_c := \{n \in \mathbb{N} \mid T^k(n) = 1, \text{ for some } k \leq c \cdot B(n)\}.$$

5.3. **Remark.** An integer that satisfies the Collatz conjecture is contained in the sets G_c, for sufficiently large c. Typically, the value of c is small. For example, all integers smaller than 10^6 are contained in G_{17}. Furthermore, all the random numbers involved in the experiments carried out are contained in the set G_5. This does not come as a surprise. If the observations about the mean and standard deviation of the total stopping time in Section 4.3 are correct, the set G_5 has density one.

5.4. **Lemma.** Let $n \in \mathbb{N}$ be an integer. Then the bit length of $T^k(n)$ is bounded by $k + B(n)$.

Proof. We have $T(x) \leq \frac{3x+1}{2} \leq 2x$, for all $x \in \mathbb{N}$. Thus, each elementary step increases the bit length by at most 1. □

5.5. **Remark.** For a fixed $c > 1$, let $n \in G_c$ be given. Then a direct check of the Collatz conjecture for n would compute $n, T(n), T(T(n)), \ldots, T^k(n) = 1$, for $k \leq c \cdot B(n)$. By Lemma 5.4, all computed numbers have at most $k + B(n) \leq (1+c) \cdot B(n)$ bits. As the number of bit operations to compute $T(x)$ is linear in $B(x)$, the direct check requires

$$O(\sum_{i=0}^{k-1} B(T^i(n))) = O(k(k + B(n))) = O(c(1+c)B(n)^2) = O_c(B(n)^2)$$

bit operations.

5.6. **Lemma.** The algorithm above computes the standard polynomial $S_{k,r}$ in $O(k \log(k)^2 \log(\log(k)))$ bit operations.

Proof. We denote by $f(k)$ the maximal number of bit operations used to compute any of the standard polynomials $\{S_{j,r} : 0 \leq r < 2^j, j \leq k\}$ and by $M(k)$ the costs to multiply two k-bit numbers. The recursive construction requires two smaller Collatz polynomials, three multiplications of numbers bounded by $\mathrm{Max}(3^{(k+1)/2}, 2^k)$, a fixed number of additions, and bit shifts. As the coefficients of $S_{k,r}$ are bounded by 3^k, we have the inequality

$$f(2k) \leq 2f(k) + 3M(k+2) + C \cdot k\,,$$

for some constants C, as the costs for addition and bit shifts are linear in the size of the input. Using $M(k) = O(k \log(k) \log(\log(k)))$ [4, Theorem 8.24], an application of [4, Lemma 8.2] shows the claim. □

5.7. **Lemma.** Let $n \in \mathbb{N}$ be given. We put $m := \mathrm{Max}(1, \lfloor \log_2(n)/2 \rfloor)$. Then the computation of $T^m(n)$ (i.e., one macro step in the algorithm above) requires $O(B(n) \log(B(n))^2 \log(\log(B(n))))$ bit operations.

Proof. The algorithm first computes the standard polynomial $S_{m, n \bmod 2^m}$. This requires $O(m \log(m)^2 \log(\log(m)))$ bit operations. Next, it evaluates the standard polynomial. As we have $m \leq B(n)/2 + 1$, the evaluation requires one multiplication of numbers with at most $B(n) + 1$ bits, an addition of numbers having at most $2B(n) + 2$ bits, and a bit shift of the result. As the cost of addition and bit shifts is linear in the size of the input, the bound above for the costs of a multiplication proves the claim. □

5.8. **Theorem.** Let $c > 1$ be a constant. Then the algorithm above verifies the Collatz conjecture for $n \in G_c$ by using

$$O_c(B(n) \log(B(n))^2 \log(\log(B(n))))$$

bit operations.

Proof. As the claim is an asymptotic statement, we assume that $\log(\log(B(n))$ is defined and at least 1. We denote by $k_1, \ldots, k_r$ the sizes of the macro steps used in the above algorithm and by $U_0 := n, U_1 := T^{k_1}(n), \ldots, U_r := T^{k_r}(U_{r-1})$ the sequence of numbers computed by the macro steps. Using Lemma 5.4 and the assumption $n \in G_c$, we can bound $B(U_i)$ by $(1+c)B(n)$. Thus, by Lemma 5.7, the number of bit operations used for the i-th macro step is bounded by

$$C \cdot B(U_i) \log((1+c)B(n))^2 \log(\log((1+c)B(n)))\,,$$

for some constant C.

Furthermore, we have

$$\sum_{i=0}^{r-1} B(U_i) \leq \sum_{i=1}^{r} 2k_i + 3 \leq 5 \sum_{i=1}^{r} k_i \leq 5 \cdot c \cdot B(n)\,.$$

It the last estimate, the assumption $n \in G_c$ was used. Combining the estimates above, we get

$$\begin{aligned} &5 \cdot c \cdot C \cdot B(n) \log((1+c)B(n))^2 \log(\log((1+c)B(n))) \\ =\, &O_c(B(n) \log(B(n))^2 \log(\log(B(n)))) \end{aligned}$$

as a upper bound for the number of bit operations to verify the Collatz conjecture. □

6. Conclusion

The investigation shows that the Collatz conjecture can be efficiently verified for random numbers with ten billion decimal digits. This does not prove the conjecture. But it does illustrate what is possible using modern computer algebra systems. Furthermore, the method described may be useful in other number-theoretic experiments.

References

[1] Barina, D.: *Convergence verification of the Collatz problem*, The Journal of Supercomputing, Volume **77**, Issue 3, (2021), 2681 – 2688

[2] Barina, D.: *Improved verification limit for the convergence of the Collatz conjecture*, The Journal of Supercomputing, Volume **81**, 810 (2025)

[3] W. Bosma, J. Cannon, C. Playoust: *The Magma algebra system.* I. The user language, J. Symbolic Comput. **24** (1997), 235 – 265.

[4] J. von zur Gathen and J. Gerhard: *Modern computer algebra*, Cambridge Univ. Press, New York, 1999

[5] T. Granlund and the GMP development team: *The GNU Multiple Precision Arithmetic Library*, `https://gmplib.org/`

[6] G. T. Leavens and M. Vermeulen: $3x+1$ *search programs*, Comput. Math. Appl. **24** (1992), no. 11, 79 – 99

[7] W. Ren, S. Li, R. Xiao and W. Bi: *Collatz Conjecture for 2^100000-1 Is True - Algorithms for Verifying Extremely Large Numbers*, in: 2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), Guangzhou, China (2018), pp. 411–416

[8] W. Ren: *Collatz Computation Sequence for Sufficient Large Integers is Random*, Preprint: `https://eprint.iacr.org/2023/648.pdf`

[9] Schönhage, A.; Strassen, V.: *Schnelle Multiplikation großer Zahlen.* Computing (Arch. Elektron. Rechnen) **7** (1971), 281 – 292.

[10] Stephens M.A.: *EDF Statistics for Goodness of Fit and Some Comparisons.* Journal of the American Statistical Association **69** (1974), 730 – 737.

[11] `https://www.reddit.com/r/Collatz/comments/13g7cb0/largest_number_ever_tested/`

School of Mathematics and Statistics, University of Syndey NSW 2006, Australia

Institut für Mathematik, Universität Würzburg, Emil-Fischer-Strasse 30, D-97074 Würzburg, Germany

Email address: `stephan.elsenhans@mathematik.uni-wuerzburg.de`

URL: `https://www.mathematik.uni-wuerzburg.de/institut/personal/elsenhans.html`