Proceedings of the 2nd International Conference on Computing Innovation and Applied Physics (CONF-CIAP 2023)
DOI: 10.54254/2753-8818/5/20230579

Algorithm speed-ups of 3x+1 convergence verification sieves

Kevin Ge

W. T. Woodson High School, 9525 Main Street, Fairfax, VA 22031, United States of America

kevinge0086@gmail.com

Abstract. The 3x + 1 Conjecture is a notorious open elementary number theory problem for its deceitful triviality. Several algorithms described in the past for the numerical verification for the convergence, one of which implemented an improvement of sieves. However, most papers focused on theoretical aspects of the sieve, and empirical results were rarely discussed. This paper presents data from implementing sieves on a relatively high-level language, as well as some helpful unmentioned rudimentary algebraic facts about sieves. Several algorithms for convergence verification of the 3x + 1 Problem had been developed since the 1970s. This paper tests an algorithm presented in a 1999 paper and explores the runtime growth to input size.

Keywords: 3x + 1 problem, collatz conjecture, sieves, number theory.

1. Introduction

The $3x + 1$ Problem, which is called the Collatz Conjecture, the Ulam Conjecture, the Kakutani Problem, the Thwaites Conjecture, the Syracuse Problem, and Hasse's Algorithm, is an elementary number theory problem that is known for its simple definition yet difficulty to prove. Though discovered by Collatz in the 1930s, this problem did not gain much attention until the 1970s [1]. Since then, numerous mathematicians had been trying to prove it, but no proof of this problem has been recognized as of now.

The $3x + 1$ Conjecture states that for the function $H(n)$, where $H(n)$ is defined as:

$$H(n) = \begin{cases} 3n + 1 & if\ n \equiv 1\ (mod\ 2) \\ \dfrac{n}{2} & if\ n \equiv 0\ (mod\ 2) \end{cases}$$

All inputs n will converge to 1 after a finite number of iterations, having reached 1, iterates would forever oscillate in the $4 \to 2 \to 1$ cycle. Since an iteration of $H(n)$ delivers an odd number to an even number, one can also consider instead using the following alternative function:

$$T(n) = \begin{cases} \dfrac{3n + 1}{2} & if\ n \equiv 1\ (mod\ 2) \\ \dfrac{n}{2} & if\ n \equiv 0\ (mod\ 2) \end{cases}$$

DOI: 10.54254/2753-8818/5/20230579

Many partial results indicate the truth of this conjecture. Jeffrey Lagarias compiled two bibliographies on this topic [2-3].

The Collatz Conjecture would be shown to be false if any of the following two cases were found: the first is a trajectory that diverges to positive infinity, or the second is another cycle other than the trivial $4 \to 2 \to 1$ cycle. Currently, none of the above cases have been found. In fact, Eliahou demonstrated in 1993 that any hypothetical cycle must have a lower bound on its length, which can be determined by the number of numbers known to follow this conjecture [4]. With an earlier computation result, he demonstrated that all non-trivial cycles must have a length of at least 17,087,915 [4]. Readers who are interested can apply method to later results for stricter lower bounds.

The numerical verification has been a prominent source of empirical evidence for the truth of this conjecture. The earliest traceable record of this practice is Dunn's verification of all numbers less than 22,882,247 in 1973 [5]. In 1992, Leavens and Vermeulen improved that record to 5.6×10^{13}[1]. Oliveira e Silva significantly improved the algorithm and further pushed the record to 20×2^{58}[6]. The latest result is from Bařina in 2020, who has verified all numbers (for the truth of this conjecture) up to 2^{68}, and the result is currently constantly updated on his website [7].

Sieves are a type of improvement that was used in the preceding convergence verification. Briefly, for a family of numbers, their convergence is guaranteed and thus does not need to be checked. Theoretical estimates of the speed-up factors of applying sieves had been provided in the past [6, 8], yet little discussion of the actual runtime had been published.

2. Sieves

In the published literature, the concept of Sieves was first described by Oliveira e Silva [9], though Leavens and Vermeulen had implemented a similar idea in their earlier 1992 paper, namely that of $4n + 1$ [1].

Before that, note that if the numbers in the target interval are checked in ascending order, then there is a noteworthy improvement that can significantly save the time of computation.

2.1. Improvement

If the number is verified in the ascending order for a convergence algorithm, then each number only needs to be checked until it reaches an iterate that is smaller than the initial value.

The speed-up factor of this improvement will be tested in Section 3, and sieves, which will be described immediately below, will be implemented on top of this improvement.

Examine any k such that $k = 4n + 1$, $n, k \in \mathbb{Z}^+$, it's easy to notice that all k must follow the following path in Table 1:

Table 1. Iterations for $4n + 1$ Case.

0	$4n + 1$
1	$12n + 4$
2	$6n + 2$
3	$3n + 1$

Since $4n$ must be an even number, $4n + 1$ must be an odd number, thus, the operation $3n + 1$ will be applied. After that the number arrives at $12n + 4$, which obviously is an even number. Therefore the $\frac{n}{2}$ path will be taken. It is easy to see that any initial values k of the form $k = 4n + 1$, $n, k \in \mathbb{Z}^+$ will reach the iterate $3n + 1$ for their corresponding n. At this point, the above arguments fail to apply, since we can no longer determine whether $3n + 1$ is odd or even; but we can tell that $3n + 1$ must be smaller than $4n + 1$ for the domain of inputs. In fact, a stronger conclusion can be made about the numbers of this form.

2.2. Theorem

Let $p = an + b$, where $0 \leq b < a$ and $n, a, b \geq 0 \in \mathbb{Z}$. Similarly, let $q = cn + d$, where $0 \leq d < c$ and $n, c, d \geq 0 \in \mathbb{Z}$. If $a < c$, then we have $p > q \Leftrightarrow (a - c)n > d - b$.

DOI: 10.54254/2753-8818/5/20230579

Proof.

$$(a-c)n > d-b$$
$$an-cn > d-b$$
$$an+b > cn+d$$

Corollary 1. $p > q \Leftrightarrow n > \frac{d-b}{a-c}$

Corollary 2. $b > d \Leftrightarrow p > q \ \forall \ n$.

Proof.

$$b > d \Leftrightarrow b-d > 0$$
$$a > c, n \geq 0 \Leftrightarrow (a-c)n \geq 0 \ \forall \ n$$
$$\Leftrightarrow (a-c)n + (b-d) \geq 0 + (b-d) > 0 \ \forall \ n$$
$$\Leftrightarrow an + b - (cn+d) > 0 \ \forall \ n$$
$$\Leftrightarrow p > q \ \forall \ n$$

As there are more numbers of this form which can be eliminated, and for all of them, it can just use the inequality from Corollary 1 to check for what values of n does it apply.

For example, for the case described above, $4n+1$ and $3n+1$, the sieve applies when $n > 0$ (Corollary 1). In the case of $n = 0$, we have the trivial loop, which is out of the testing range. Therefore, this sieve can be used without any restrictions.

However, it is important to note that for the above inequality to apply, the lower number in the sieves must be of the described $cn + d$ form. Oliveira e Silva had shown that all sieves indeed comply with this property [8].

We can then proceed to generate more sieves of this kind. Oliveira e Silva provided statistical data about the number of sieves at each pruning level less than 40, as well as the theoretical speed-up gained by each depth [6,8]. Moreover, Roosendaal's website also contains relevant information [9]. In this paper, we will only apply several sieves, and instead of testing their cumulative speed-up for each level, we will record the individual cumulative speed-ups of each sieve, applied in an ascending order.

3. Application of speed-ups on object oriented language

3.1. Implementation

As it shows in the last section, initial values of the form of a sieve do not need to be tested. This section presents data on the implementation of sieves, and their recorded speed-ups when being implemented in Java, as shown in Tables 2 - 4.

Table 2. Sieve 1.

0	$2n+0$
1	$n+0$

for $n > 0$.

Table 3. Sieve 2.

0	$4n+1$
1	$12n+4$
2	$6n+2$
3	$3n+1$

for $n > 0$.

DOI: 10.54254/2753-8818/5/20230579

Table 4. Sieve 3.

0	$16n + 3$
1	$48n + 10$
2	$24n + 5$
3	$72n + 16$
4	$36n + 8$
5	$18n + 4$
6	$9n + 2$

for $n > 0$.

Notice that the inequality acquired by Corollary 1 is $n > -\frac{1}{7}$, but we can simplify it to $n > 0$ due to the domain of n. Alternatively, we can also use Corollary 2 to arrive at the equivalent conclusion that this sieve applies to all n. Inequalities like this will be simplified similarly later in this article, as shown in Tables 5 and 6.

Table 5. Sieve 4.

0	$32n + 11$
1	$96n + 34$
2	$48n + 17$
3	$144n + 52$
4	$72n + 26$
5	$36n + 13$
6	$108n + 40$
7	$54n + 20$
8	$27n + 10$

for $n > 0$.

Table 6. Sieve 5.

0	$32n + 23$
1	$96n + 70$
2	$48n + 35$
3	$144n + 106$
4	$72n + 53$
5	$216n + 160$
6	$108n + 80$
7	$54n + 40$
8	$27n + 20$

for $n > 0$.

The convergence algorithm was tested in different scenarios and the average runtime out of 15 trials are recorded. Table 6 gives a comparison between the runtime when Improvement 1 (to apply $T(n)$ to the input only until it reaches an iterate less than itself) is and is not implemented. Table 7 presents comparison for runtimes when the 5 mentioned Sieves are applied cumulatively in the ascending order of their corresponding a value. That is, the runtime when only Sieve 1 is implemented, the runtime when both Sieve 1 and Sieve 2 are implemented, etc. Table 8 gives complementary speedup factors to Table 7.

The “Range Tested” column refer to the range of number starting from 3 to 1 below the number in the corresponding row. For example, “1,000” refer to the range from 3 to 999. 1 and 2 do not to be tested because they are a part of the trivial cycle.

DOI: 10.54254/2753-8818/5/20230579

3.2. Result

We can see from the Table 7 that there is a slight decline in the speed-up factor of Improvement 1 at 10,000. This is expected at small initial values due to the time that was spent on checking whether the input is of the form of a sieve or not being more than the amount of time that would have been saved from checking these numbers. The speed-up factor then proceeds to grow monotonically for the corresponding ranges of 10,000 to 100,000,000. This agrees with the prediction, as the time skipped checking sieve values grows much faster than the time checking if a number is a sieve or not. However, the growth slows down after 1,000,000. This could indicate certain properties of the growth of cumulative stopping time, as the time checking if the input is of the form of a sieve or not grows linearly.

Table 8 presents data about speed-up factor of implementing sieves for increasing ranges based on data in Table 7. For small ranges, the speed-up factor is somewhat random, sometimes even negative. However, for larger ranges, they tend to stay positive and grow monotonically. For instance, the speed up factor remained positive 10,000,000 and 100,000,000, but fluctuates between positive and negative for the 3 other rows. The growth, however, is random for all ranges. An interesting question one could ask is whether changing the order which sieves are applied affect the final speed up. For the purpose of this paper, this topic will not be discussed here.

Form the Tables 7-9, we give the following suggestion to the implementation of sieves in verification algorithms. Improvement 1 should be implemented regardless of the range. Sieves, however, should be implemented according to the range and the magnitude of values tested. Specifically, for small ranges or small values, implementing sieves of rare occurrences (those with a higher a value) might slow down the program, for the time used to check if a number is of the form of a sieve can very easily overwhelm the time reduced for verifying the sieves. Empirically, for the 5 sieves presented, we recommend the reader to implement them for positive values in Table 8 only.

Table 7. Average runtime (milliseconds).

Range Tested	No Sieves	With Improvement 1	Speed-up
1,000	16.2	9.5	41.36%
10,000	64.2	39.1	39.10%
100,000	330.7	136.1	58.84%
1,000,000	3293.4	584.7	82.25%
10,000,000	42402.7	3305.5	92.20%
100,000,000	632096.4	31832.3	94.96%

Table 8. Average runtime (milliseconds) with cumulative sieves.

Range Tested	No Sieves	Sieve 1	Sieve 2	Sieve 3	Sieve 4	Sieve 5
1,000	9.5	7.9	6.5	7.5	4.3	3.4
10,000	39.1	29.6	24.0	27.0	13.5	12.9
100,000	136.1	111.1	83.7	86.7	50.7	47.7
1,000,000	584.7	458.1	395.7	437.1	239.6	242.5
10,000,000	3305.5	2526.5	2063.2	1578.3	1129.0	1125.6
100,000,000	31832.3	24707.8	20041.7	14907.5	10069.1	9814.9

Table 9. Speed-ups of cumulative sieves.

Range Tested	Sieve 1	Sieve 2	Sieve 3	Sieve 4	Sieve 5
1,000	16.84%	17.72%	-15.38%	42.67%	20.93%
10,000	18.92%	18.92%	-12.50%	50.00%	4.44%
100,000	18.37%	24.66%	-3.58%	41.52%	5.92%
1,000,000	21.65%	13.62%	-10.46%	45.18%	-1.21%
10,000,000	23.57%	18.34%	23.50%	28.47%	0.41%
100,000,000	22.38%	18.89%	25.62%	32.46%	2.52%

DOI: 10.54254/2753-8818/5/20230579

4. Conclusion

This paper takes a deeper look at the basic algebraic properties of sieves and integers of a similar form. This paper also analyzes the application speed-up of sieves being implemented consecutively. Different from many predecessors on this topic, this paper analyzes the speed-up each individual sieve brings when they are added instead of adding multiple sieves of the same depth. This paper also provides a guide to the choice of implementation in relatively object-oriented high-level languages. The results confirm with the prediction, which highlights the fact that sieves are more efficient to be implemented when the range tested, or the average input number is of a greater magnitude. A reference of speed-ups of ranges starting at 3 and ending at various powers of 10 was provided, which can be used as a guide to produce a more general table regarding the relationship between runtime, range and starting values in the range.

References

[1] Leavens, G. T., & Vermeulen, M. (1992). 3x+ 1 search programs. Computers & Mathematics with Applications, 24(11), 79-99.
[2] Lagarias, J. C. (2003). The 3x+ 1 problem: An annotated bibliography (1963–1999). The ultimate challenge: the 3x, 1, 267-341.
[3] Lagarias, J. C. (2006). The 3x+ 1 problem: An annotated bibliography, II (2000-2009). arXiv preprint math/0608208.
[4] Eliahou, S. (1993). The 3x+ 1 problem: new lower bounds on nontrivial cycle lengths. Discrete mathematics, 118(1-3), 45-56.
[5] Dunn R (1973) On Ulam's problem. Tech. rep., University of Colorado at Boulder.
[6] Oliveira e Silva, T. (2010). Empirical verification of the 3x+1 and related conjectures. The ultimate challenge: The 3x+1 problem, 189-207.
[7] Bařina, D. (2021). Convergence verification of the Collatz problem. The Journal of Supercomputing, 77(3), 2681-2688. doi: 10. 1007 / s11227 - 020 - 03368-x.
[8] Bařina, D. (2021). url: https://pcbarina.fit.vutbr.cz/.
[9] Oliveira e Silva, T. (1999). Maximum excursion and stopping time record-holders for the 3x+ 1 problem: computational results. Mathematics of Computation, 68(225), 371-384.
[10] Roosendaal, E., On the 3x + 1 problem, url: http://www.ericr.nl/wondrous/index.html.