MATHEMATICS OF COMPUTATION
Volume 68, Number 225, January 1999, Pages 371–384
S 0025-5718(99)01031-5

MAXIMUM EXCURSION AND STOPPING TIME RECORD-HOLDERS FOR THE $3x+1$ PROBLEM: COMPUTATIONAL RESULTS

TOMÁS OLIVEIRA E SILVA

ABSTRACT. This paper presents some results concerning the search for initial values to the so-called $3x+1$ problem which give rise either to function iterates that attain a maximum value higher than all function iterates for all smaller initial values, or which have a stopping time higher than those of all smaller initial values. Our computational results suggest that for an initial value of n, the maximum value of the function iterates is bounded from above by $n^2 f(n)$, with $f(n)$ either a constant or a very slowly increasing function of n. As a by-product of this (exhaustive) search, which was performed up to $n = 3 \cdot 2^{53} \approx 2.702 \cdot 10^{16}$, the $3x+1$ conjecture was verified up to that same number.

1. INTRODUCTION

The $3x+1$ problem, also known as the Collatz problem, is concerned with the behavior of the iterates of the function $T(n) : \mathbb{N}_0 \to \mathbb{N}_0$ defined by

$$T(n) = \begin{cases} n/2 & \text{if } n \text{ is even,} \\ (3n+1)/2 & \text{if } n \text{ is odd.} \end{cases}$$

($\mathbb{N}_0$ denotes the set of non-negative integers.) The $3x+1$ conjecture asserts that repeated iteration of $T(n)$, starting from any positive integer n, eventually produces the value 1 [3]. This easily stated conjecture has not yet been proved (or disproved). To prove it, one has to prove that the iterates[1] $T^{(k)}(n)$ of $T(n)$ remain bounded for each $n > 0$, and that the only solutions in the positive integers of the equation $n = T^{(k)}(n)$ are k even, and $n = 1$ or $n = 2$. A very good account on the history of the $3x+1$ problem, as well as a large collection of known results and conjectures related to it, can be found in [3]. We will follow the notation and terminology of that paper whenever possible.

The trajectory of n is the sequence $\left(T^{(k)}(n)\right)_{k=0}^{\infty}$. The stopping time of n, denoted by $\sigma(n)$, is the least positive k for which $T^{(k)}(n) < n$, if it exists, or infinity, otherwise. The maximum excursion of the trajectory of n, denoted by $t(n)$, is the maximum value of $T^{(k)}(n)$ for $k \geq 0$, if it exists, or infinity, otherwise. We call $n > 1$ a σ-record-holder if $\sigma(m) < \sigma(n)$ for all $1 < m < n$, i.e., if all integers smaller than n (and larger than 1) have smaller stopping times. Similarly, we call

Received by the editor January 3, 1997.
1991 *Mathematics Subject Classification.* Primary 26A18; Secondary 11Y99.
Key words and phrases. $3x+1$ problem, Collatz problem, algorithm, search, $3x+1$ conjecture.

[1] $T^{(k)}(n)$ denotes the k-th iterate of $T(n)$: $T^{(0)}(n) = n$, $T^{(1)}(n) = T(n)$, and, for $k > 0$, $T^{(k)}(n) = T(T^{k-1}(n))$.

n a t-record-holder if $t(m) < t(n)$ for all $1 < m < n$. The main objective of this paper is to find all t-record-holders in the interval $1 < n < 3 \cdot 2^{53}$, with the purpose of studying empirically the rate of growth of $t(n)$. As a side result, we also check the $3x+1$ conjecture, and find all σ-record-holders, in the same range. The results of a previous search, conducted by Leavens and Vermeulen, broader in scope but on a smaller range ($5.6 \cdot 10^{13}$), are reported in [4].

Equipped with these definitions, the $3x+1$ conjecture states that every integer $n \geq 2$ has a finite stopping time (which implies that $t(n) < \infty$ for all $n \geq 2$). This alternative formulation of the $3x+1$ conjecture rests on the observation that if it is known that all trajectories for $n < n_0$ reach the value 1 after a finite number of iterates, then, to prove the same for n_0, it is enough to iterate T at n_0 until $T^{(k)}(n_0) < n_0$, i.e., up to $k = \sigma(n_0)$.

The rest of this paper is organized as follows. In section 2 we describe some interesting facts about the structure of the iterates of $T(n)$. These facts are exploited in section 3, and lead to the construction of an efficient algorithm to search exhaustively for t- and σ-record-holders. The results of this exhaustive search are reported in section 4, and provide numerical evidence for a conjecture, stated in section 5, concerning the growth of $t(n)$. Subsection 4.1 provides a theoretical analysis of the speedup of the proposed search algorithm over a "naive" search algorithm.

Unless explicitly stated otherwise, the theoretical results reported in this paper were (re)discovered by the author. Quite probably, most of them, if not all, are common knowledge in the $3x+1$ research community.

2. On the structure of the iterates of $T(n)$

Our first task is to show that it is possible to determine, using only the last k base-2 digits of n_0, which branches of $T(n)$ will be taken in the computation of $T^{(k)}(n_0)$. For $k = 1$ this is an immediate consequence of the definition of $T(n)$, and is put is evidence by setting $n_0 = 2n_1 + r_0$, where r_0 is the last base-2 digit of n_0. In fact, applying $T(n)$ to n_0 gives

$$T(n_0) = T(2n_1 + r_0) = \begin{cases} n_1 & \text{if } r_0 = 0, \\ 3n_1 + 2 & \text{if } r_0 = 1. \end{cases}$$

It is also clear that we can compute $T^{(2)}(n_0)$ directly, by setting $n_1 = 2n_2 + r_1$, where r_1 is the last base-2 digit of n_1 (and therefore the next to last base-2 digit of n_0). A few simple computations yield

$$T^{(2)}(n_0) = \begin{cases} n_2 & \text{if } r_0 = 0 \text{ and } r_1 = 0, \text{ i.e., if } n_0 = 4n_2 + 0, \\ 3n_2 + 2 & \text{if } r_0 = 0 \text{ and } r_1 = 1, \text{ i.e., if } n_0 = 4n_2 + 2, \\ 3n_2 + 1 & \text{if } r_0 = 1 \text{ and } r_1 = 0, \text{ i.e., if } n_0 = 4n_2 + 1, \\ 9n_2 + 8 & \text{if } r_0 = 1 \text{ and } r_1 = 1, \text{ i.e., if } n_0 = 4n_2 + 3. \end{cases}$$

It is possible to proceed in the same fashion to compute $T^{(k)}(n_0)$ for all positive k. The fate of each new iterate requires the knowledge of one more base-2 digit of n_0. Let $n_0 = 2^k n_k + m_k$, with[2] $n_k = \lfloor n_0/2^k \rfloor$ and $m_k = n_0 \bmod 2^k$. We claim that

Property 1. *The general form of $T^{(k)}(n)$ is*

$$T^{(k)}(n_0) = 3^{y(k;n_0)} n_k + T^{(k)}(m_k), \qquad k \geq 0, \tag{1}$$

[2] $\lfloor x \rfloor$ denotes the largest integer smaller than or equal to x.

where $y(k;n_0)$ is the number of odd branches of $T(n)$ that were taken in the computation of $T^{(k)}(n_0)$ [or of $T^{(k)}(m_k)$].

This result is clearly true for $k = 0$, 1, and 2. For the general case, it can be proven easily by induction, with the help of the formulas $n_k = 2n_{k+1} + r_k$ and $m_{k+1} = r_k 2^k + m_k$, where r_k is the last base-2 digit of n_k. In fact, applying these formulas to (1), and also using (1) to simplify the result, gives

$$T^{(k)}(n_0) = 2 \cdot 3^{y(k;n_0)} n_{k+1} + T^{(k)}(m_{k+1}). \tag{2}$$

The computation of $T\big(T^{(k)}(n_0)\big)$ then shows that if (1) is valid up to a certain k then it is also valid for $k+1$, thereby proving our claim. Note that

$$y(k+1;n_0) = y(k+1;m_{k+1}) = \begin{cases} y(k;n_0) & \text{if } T^{(k)}(m_{k+1}) \text{ is even,} \\ y(k;n_0)+1 & \text{if } T^{(k)}(m_{k+1}) \text{ is odd.} \end{cases}$$

Our next task is to show that the result of the k-th iteration of $T(n)$ can be naturally expressed in base-3, as suggested by (1). We claim that

Property 2. *For $k \geq 0$,*

$$T^{(k)}(m_k) < 3^{y(k;m_k)}. \tag{3}$$

That is, $T^{(k)}(m_k)$, when expressed in base-3, has no more that $y(k;m_k)$ digits. Again, this is clearly true for $k = 0$, 1, and 2. Once again, the general case can be proven easily by induction. Assuming that (3) is true up to a certain k allows us to conclude that

$$T^{(k)}(m_{k+1}) = 3^{y(k;n_0)} r_k + T^{(k)}(m_k) < 2 \cdot 3^{y(k;n_0)}.$$

It is then a simple matter to verify that (3) is also true for $k+1$, proving our claim. In words, properties 1 and 2 state that the representation of $T^{(k)}(n_0)$ in base-3 is very simple: the last $y(k;n_0)$ digits are given by $T^{(k)}(m_k)$, and the rest by n_k [cf. (1) and (3)]. This is illustrated in Figure 1, where it is shown pictorially that the computations leading to $T^{(k)}(n_0)$, for an arbitrary non-negative n_0, are naturally organized in a binary tree.

Our final task is to explain some additional structure that we have noticed in the iterates of $T(n)$. Generalizing the ideas that led to (2), it is not difficult to verify that

Property 3. *For $0 \leq p \leq k$,*

$$T^{(p)}(n_0) = 2^{k-p} \cdot 3^{y(p;m_k)} n_k + T^{(p)}(m_k),$$

and

$$T^{(p)}(m_k) < 2^{k-p} \cdot 3^{y(p;m_k)} n_k.$$

Concerning the relative order of the iterates of $T(n)$, we have observed numerically that

Property 4. *For $1 \leq k \leq 40$ and $1 < m_k < 2^k$, if we sort the numbers*

$$2^{k-p} \cdot 3^{y(p;m_k)}, \qquad 0 \leq p \leq \min\{k, \sigma(m_k)\},$$

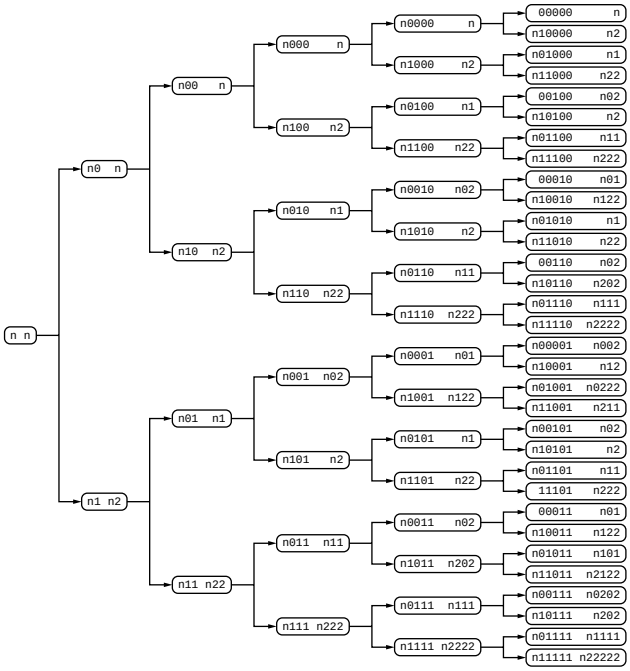

FIGURE 1. This tree describes the results of iterating $T(n)$ k-times, $0 \le k \le 5$. The node depth is equal to the number of iterates of $T(n)$. Inside each node (oval box), the number in the left-hand side is n_0 (expressed in base-2), and the number in the right-hand side is $T^{(k)}(n_0)$ (expressed in base-3). To conserve space, inside each node n_k was replaced by just n.

in increasing order, then the corresponding numbers

$$T^{(p)}(m_k), \qquad 0 \le p \le \min\{k, \sigma(m_k)\},$$

also become sorted in increasing order.[3]

This property implies that

Property 5. *If $1 \le k \le 40$, $1 < m_k < 2^k$, and $n_k \ge 0$, then the relative order of the numbers $T^{(p)}(m_k)$ is the same as that of the numbers $T^{(p)}(2^k \cdot n_k + m_k)$ for $0 \le p \le \min\{k, \sigma(m_k)\}$.*

This implies that the maximum (and the minimum) of the first $k \le 40$ iterates occurs, for each $1 < m_k < 2^k$ and for all $n_k \ge 0$, always in the same iterate number. A simple consequence of property 5, which follows from the comparison of the p-th iterate of $2^k \cdot n_k + m_k$ with the 0-th iterate, is

Property 6. *Let $\sigma(m_k) = p \le k \le 40$ and $1 < m_k < 2^k$. Then, the stopping time of the trajectory of $2^k \cdot n_k + m_k$, for any $n_k \ge 0$, is exactly equal to p.*

[3]To reduce the computational effort, we have restricted the verification of this property to the upper bound $\min\{k, \sigma(m_k)\}$ and to $k \le 40$. (The reason for this will become apparent later on.) We conjecture that it remains true for $k > 40$, and that it can be extended to $p > \sigma(m_k)$, as long as $T^{(p)}(m_k) > 1$.

The two congruence classes not treated in property 6, viz. $m_k = 0$ and $m_k = 1$, are special. The first one gives rise either to a cycle of period one, if $k = 1$ and $n_k = 0$, or to a trajectory with a stopping time equal to one, if $k > 0$ and $n_k > 0$ [recall that $T(2n) = n$]. The second one gives rise either to a cycle of period two, if $k = 2$ and $n_k = 0$, or to a trajectory with a stopping time equal to two, if $k > 1$ and $n_k > 0$ [recall that $T^{(2)}(4n + 1) = 3n + 1$]. We then have

Property 7. *The stopping time of the trajectory of* $2n$, $n > 0$, *is one, and the stopping time of the trajectory of* $4n + 1$, $n > 0$, *is two.*

Note that properties 4, 6, and 7, taken together, confirm the coefficient stopping time conjecture [3, p. 11] for stopping times smaller than 41.

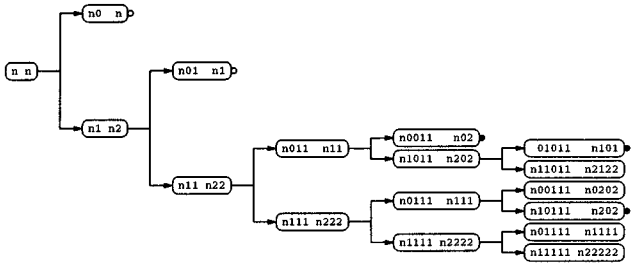

FIGURE 2. The search tree for $0 \le k \le 5$. A small open circle ($\circ$) after a node indicates a cycle (for $n_k = 0$). A small shaded circle ($\bullet$) after a node indicates that the stopping time for the congruence class of initial values associated with that node is exactly equal to that node's depth. These two types of nodes do not need to be subdivided. Only the nodes with maximal depth (and without a circle) are used in the search.

3. The exhaustive search strategy

Properties 6 and 7 are fundamental to our exhaustive search strategy. They allow us to discard the offspring of all nodes of Figure 1 that satisfy $T^{(k)}(m_k) \le m_k$. All nodes of the resultant (pruned) tree, which is depicted in Figure 2, are associated with one of three cases:

1. The last iterate attained, for the first time, a value lower than the initial value of the trajectory (the node's depth is the stopping time for all initial trajectory values belonging to the congruence class associated with that node). These nodes are marked with a shaded circle, and do not need to be subdivided.
2. The last iterate reached, for $n_k = 0$ (k is the depth of the node), a value equal to the initial value of the trajectory (a cycle). These nodes are marked with an open circle in Figure 2, and, for $n_k > 0$, they behave as the nodes marked with a shaded circle. Hence, they also do not need to be subdivided.
3. The maximal depth used in the tree construction was reached. The stopping time for all initial trajectory values belonging to the congruence class associated with nodes of this kind is larger that the maximal tree depth.

The nodes of type 1 or 2 above will be called closed nodes. All other nodes of the tree will be called open nodes. The number of open and closed nodes at each depth of the tree, denoted respectively by $n_o(k)$ and $n_c(k)$, up to depth 40, are presented

TABLE 1. Number of open and closed nodes at depth k

k	$n_{\mathrm{o}}(k)$	$n_{\mathrm{c}}(k)$	$n_{\mathrm{o}}(k)/2^k$	$n_{\mathrm{c}}(k)/2^k$
0	1	0	1.000 000	0.0
1	1	1	0.500 000	0.500 000
2	1	1	0.250 000	0.250 000
3	2	0	0.250 000	0.0
4	3	1	0.187 500	0.062 500
5	4	2	0.125 000	0.062 500
6	8	0	0.125 000	0.0
7	13	3	0.101 563	0.023 438
8	19	7	0.074 219	0.027 344
9	38	0	0.074 219	0.0
10	64	12	0.062 500	0.011 719
11	128	0	0.062 500	0.0
12	226	30	0.055 176	0.007 324
13	367	85	0.044 800	0.010 376
14	734	0	0.044 800	0.0
15	1 295	173	0.039 520	0.005 280
16	2 114	476	0.032 257	0.007 263
17	4 228	0	0.032 257	0.0
18	7 495	961	0.028 591	0.003 666
19	14 990	0	0.028 591	0.0
20	27 328	2 652	0.026 062	0.002 529
21	46 611	8 045	0.022 226	0.003 836
22	93 222	0	0.022 226	0.0
23	168 807	17 637	0.020 123	0.002 102
24	286 581	51 033	0.017 082	0.003 042
25	573 162	0	0.017 082	0.0
26	1 037 374	108 950	0.015 458	0.001 623
27	1 762 293	312 455	0.013 130	0.002 328
28	3 524 586	0	0.013 130	0.0
29	6 385 637	663 535	0.011 894	0.001 236
30	12 771 274	0	0.011 894	0.0
31	23 642 078	1 900 470	0.011 009	0.000 885
32	41 347 483	5 936 673	0.009 627	0.001 382
33	82 694 966	0	0.009 627	0.0
34	151 917 636	13 472 296	0.008 843	0.000 784
35	263 841 377	39 993 895	0.007 679	0.001 164
36	527 682 754	0	0.007 679	0.0
37	967 378 591	87 986 917	0.007 039	0.000 640
38	1 934 757 182	0	0.007 039	0.0
39	3 611 535 862	257 978 502	0.006 569	0.000 469
40	6 402 835 000	820 236 724	0.005 823	0.000 746

in Table 1. In that table, we also present the ratios $n_{\mathrm{o}}(k)/2^k$ and $n_{\mathrm{c}}(k)/2^k$. The latter gives us the densities of the trajectories with stopping time exactly equal to k. The former gives us the density of trajectories with stopping time higher than k [in the notation of [3], this density is given by $1 - F(k)$]. In the appendix, we present a set of recurrence formulas to compute $n_{\mathrm{o}}(k)$ and $n_{\mathrm{c}}(k)$. It produces correct results provided property 4 is true up to (and including) k. In Figure 3 we depict the graph of the base-10 logarithm of $n_{\mathrm{o}}(k)/2^k = 1 - F(k)$ as a function of k, for $k \leq 100$.

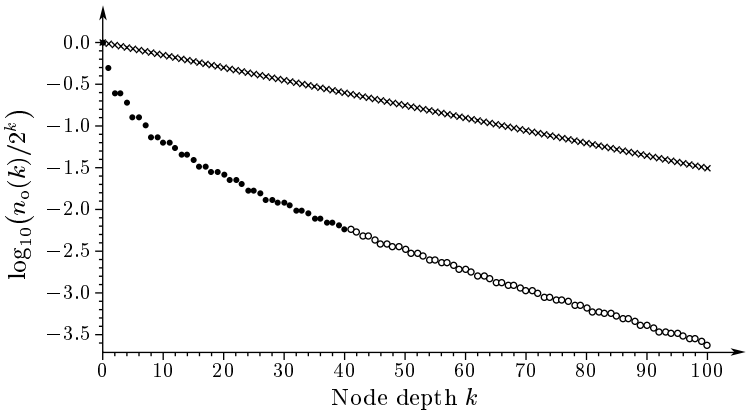

FIGURE 3. This linear–log graph depicts the ratio between open nodes at depth k of the search tree, and the total number of nodes (2^k) at that same depth. Shaded circles (•) represent exact values (generated by our program), open circles (○) represent values computed with the recurrence formulas described in the appendix, and crosses (×) represent the upper bound of $1 - F(k)$ given by Theorem D of [3].

Our search strategy is to analyze only initial values that belong to congruence classes corresponding to nodes of type 3. As Table 1 and Figure 3 demonstrate, to reduce the density of these nodes, thus speeding up the search, one should use a maximal tree depth as large as possible. In our program we have used a maximal depth of 40. This depth is the largest one that can be used in the construction of a pruned tree similar to that of Figure 2, without taking precautions against arithmetic overflows in a machine with 64-bit registers (DEC Alpha). Since a maximal depth this large gives rise to an enormous number of type 3 nodes — 6 402 835 000 to be exact — it is impractical to store and sort them in increasing order of their corresponding congruence classes. This makes a sequential search infeasible.[4] Instead, our program processes the congruence classes in the order in which they are generated. Although this makes the program somewhat more complicated, this method has the important advantage that it requires an insignificant amount of memory.

To cope with the nonsequential search, the program maintains two lists, sorted by the initial value of the trajectories: one for candidate t-record-holders, and another for candidate σ-record-holders. To reduce drastically the overhead of managing these two lists, the program analyzes 2^{10} "consecutive" initial values belonging to the congruence class of each type 3 node. Since these 2^{10} values are tested in increasing order, only one pass (per node) through these lists is required to update them. In normal circumstances, these lists have less that 80 and 40 elements, respectively.

Each type 3 node has associated with it three numbers: m_{40}; $T^{(40)}(m_{40})$, which is larger than m_{40}; and $y(40; m_{40})$. Since these numbers are available, it is sensible

[4]The search reported in [4] appears to use a sequential search, making use, among other things, of a (sorted) pruned tree of depth 16.

to put them to good use. We do that by using property 1 to compute with very little overhead $T^{(40)}(n_{40} \cdot 2^{40} + m_{40})$, thus avoiding the initial 40 iterates of $T(n)$. Note that, since the 2^{10} values of n_{40} are consecutive, we only need to apply (1) for the first one, which requires one time-expensive multi-precision multiplication and one time-inexpensive multi-precision addition. The remaining $2^{10} - 1$ values only require one (inexpensive) multi-precision addition each.

Our search strategy introduces two problems. The first one is that σ-record-holders with stopping times smaller than 41 are not reported. Since $\sigma(27) = 59 > 40$, the missing ones can be found very easily by testing all initial values between 2 and 27. A more serious problem occurs for t-record-holders. It is quite expensive to compute, for each type 3 node, the maximum of the first 40 iterates for its congruence class (recall property 5). Since that information is important only for small initial values, we have not done it. This has the implication that the maximum excursion information for small initial values may be inaccurate, if that maximum excursion occurs before the fortieth iterate. To generate the missing t-record-holders, and to remove the wrong ones, we, *a posteriori*, analyzed the results of the search. Let c_k be the k-th candidate t-record-holder. Using the simple bound

$$T^{(k)}(n) \le \frac{3^k - 1}{2^k - 1}\, n,$$

it is clear that if

$$\frac{3^{40} - 1}{2^{40} - 1}\, c_{k+1} < t(c_k)$$

then there can be no t-record-holders between c_k and c_{k+1} with a maximum excursion occurring in the first 40 iterates. Based on this observation, we traversed the candidate t-record-holders list looking for the last candidate that did not satisfy this condition. We then analyzed all initial values smaller than the candidate immediately following the last one found. It turned out that we had to analyze all initial values smaller than 319 804 831. (As it happens, this number is rather conservative. The largest t-record-holder with a maximum excursion occurring before the fortieth iterate is 704 511.)

There is one final (and important) optimization that did not occur to us. We found the idea, in a form more general than the one described here, in [4]. The idea is to ignore initial values that belong to the trajectory of a smaller initial value (this is a special case of the so-called trajectory coalescence). Consider, for example, the trajectory of $2n + 1$, which is $(2n + 1,\ 3n + 2,\ \dots)$. Since $3n + 2 > 2n + 1$ for all $n \ge 0$, it is clear that $3n + 2$ cannot be a t- or a σ-record-holder. It is therefore not necessary to analyze initial values congruent to 2 mod 3. The next interesting case is based on the relation $T^{(3)}(8n + 3) = 9n + 4$, which renders the analysis of initial values congruent to 4 mod 9 unnecessary. Other cases of this sort can be found by iterating several times the inverse function $T^{-1}(n)$, with an initial value belonging to each one of the congruence classes modulo 3^k, and looking for an inverse iterate smaller that the initial value. The congruence classes for which this happens will be called closed. Table 2 presents the number of closed congruence classes modulo 3^k for $1 \le k \le 10$. Unfortunately, the percentage of closed congruence classes increases very slowly with k. Thus, there is hardly any advantage in using congruent classes modulo 3^k with a "large" k. In our program we have used $k = 2$, which excludes 4/9 of all initial values, and requires a very small amount of memory to implement.

TABLE 2. Number of closed congruence classes modulo 3^k

k	$n(k)$	$n(k)/3^k$
1	1	33.333%
2	4	44.444%
3	12	44.444%
4	37	45.679%
5	111	45.679%
6	335	45.953%
7	1 013	46.319%
8	3 039	46.319%
9	9 145	46.461%
10	27 435	46.461%

4. Computational results

We have coded a program that implements all the techniques described in the previous section. All multi-precision arithmetic was performed on numbers stored in two 64-bit registers. The program was coded entirely in `C`. The function $T(n)$ was implemented with a multi-precision version of the code

```
if(n & 1)
  n += (n >> 1) + 1;
else
  n >>= 1;
```

which avoids a time consuming multiplication by 3. The test to update the maximum excursion was placed inside the `if` part of the code, together with a test to detect arithmetic overflows. The stopping condition for the iterations was $T^{(k)}(n) < n$, and was placed inside the `else` part of the code. Note that the existence of a cycle would put the program into an endless loop, which did not happen.

It turned out that the idea of using (1) to do several iterates of $T(n)$ in one step did not bring any speed advantage. This was due to the time-consuming multi-precision multiplication it requires, to the large amount of memory required to implement it, to the extra time required to test for a possible maximum excursion, and to the extra time required to test the stopping condition $T^{(k)}(n) < n$. Remarkably, the same idea was used with success in [4], being responsible for a program speedup factor of 7.

One important feature of our program is that its working set ("active" code and data) fits nicely into the processor's on-chip caches. Each run of the program tests exhaustively an interval of 2^{40+10} initial values, and takes around 6 CPU weeks on a DEC Alpha 266MHz computer. We have used 4 DEC Alpha computers during the search, two 266MHz models and two older 133MHz models, each one testing a different range. The results of the search are presented in Tables 3 and 4. These results, up to $5.6 \cdot 10^{13}$ for t-record-holders, and up to $6.8 \cdot 10^{12}$ for σ-record-holders, are in perfect accord with the results reported in [4].

For the record, our search program tests, on the average, an *interval* of about 317 million integers each second (on a DEC Alpha 266MHz machine, and with the program written entirely in `C`).

TABLE 3. List of all t-record-holders up to $3 \cdot 2^{53} \approx 2.702 \cdot 10^{16}$

n	$t(n)$	$t(n)/n^2$
2	2	0.500
3	8	0.889
7	26	0.531
15	80	0.356
27	4 616	6.332
255	6 560	0.101
447	19 682	0.099
639	20 762	0.051
703	125 252	0.253
1 819	638 468	0.193
4 255	3 405 068	0.188
4 591	4 076 810	0.193
9 663	13 557 212	0.145
20 895	25 071 632	0.057
26 623	53 179 010	0.075
31 911	60 506 432	0.059
60 975	296 639 576	0.080
77 671	785 412 368	0.130
113 383	1 241 055 674	0.097
138 367	1 399 161 680	0.073
159 487	8 601 188 876	0.338
270 271	12 324 038 948	0.169
665 215	26 241 642 656	0.059
704 511	28 495 741 760	0.057
1 042 431	45 119 577 824	0.042
1 212 415	69 823 368 404	0.048
1 441 407	75 814 787 186	0.036
1 875 711	77 952 174 848	0.022
1 988 859	78 457 189 112	0.020
2 643 183	95 229 909 242	0.014
2 684 647	176 308 906 472	0.024
3 041 127	311 358 950 810	0.034
3 873 535	429 277 584 788	0.029
4 637 979	659 401 147 466	0.031
5 656 191	1 206 246 808 304	0.038
6 416 623	2 399 998 472 684	0.058
6 631 675	30 171 305 459 816	0.686
19 638 399	153 148 462 601 876	0.397
38 595 583	237 318 849 425 546	0.159
80 049 391	1 092 571 914 585 050	0.171
120 080 895	1 638 950 788 059 290	0.114
210 964 383	3 202 398 580 560 632	0.072
319 804 831	707 118 223 359 971 240	6.914
1 410 123 943	3 562 942 561 397 226 080	1.792
8 528 817 511	9 072 297 468 678 299 012	0.125
12 327 829 503	10 361 199 457 202 525 864	0.068
23 035 537 407	34 419 078 320 774 113 520	0.065
45 871 962 271	41 170 824 451 011 417 002	0.020
51 739 336 447	57 319 808 570 806 999 220	0.021
59 152 641 055	75 749 682 531 195 100 772	0.022
59 436 135 663	102 868 194 685 920 926 084	0.029
70 141 259 775	210 483 556 894 194 914 852	0.043
77 566 362 559	458 306 514 538 433 899 928	0.076
110 243 094 271	686 226 824 783 134 190 180	0.056
204 430 613 247	707 630 396 504 827 495 544	0.017
231 913 730 799	1 095 171 911 941 437 256 778	0.020
272 025 660 543	10 974 241 817 835 208 981 874	0.148
446 559 217 279	19 766 638 455 389 030 190 536	0.099
567 839 862 631	50 270 086 612 792 993 117 994	0.156
871 673 828 443	200 279 370 410 625 061 016 864	0.264
2 674 309 547 647	385 209 974 924 871 186 526 136	0.054
3 716 509 988 199	103 968 231 672 274 974 522 437 732	7.527
9 016 346 070 511	126 114 763 591 721 667 597 212 096	1.551
64 848 224 337 147	637 053 460 104 079 232 893 133 864	0.151
116 050 121 715 711	1 265 292 033 916 892 480 613 118 196	0.094
201 321 227 677 935	2 636 975 512 088 803 001 946 985 208	0.065
265 078 413 377 535	2 857 204 078 078 966 555 847 716 826	0.041
291 732 129 855 135	3 537 558 936 133 726 760 243 328 464	0.042
394 491 988 532 895	6 054 282 113 227 445 504 606 919 650	0.039
406 738 920 960 667	12 800 696 705 021 228 411 442 619 682	0.077
613 450 176 662 511	22 881 441 742 972 862 145 992 619 776	0.061
737 482 236 053 119	37 684 665 798 782 446 690 107 505 928	0.069
1 254 251 874 774 375	1 823 036 311 464 280 263 720 932 141 024	1.159
5 323 048 232 813 247	1 964 730 439 297 455 725 829 478 995 944	0.069
8 562 235 014 026 655	13 471 057 008 351 679 202 003 944 688 336	0.184
10 709 980 568 908 647	175 294 593 968 539 094 415 936 960 141 122	1.528

TABLE 4. List of all σ-record-holders up to $3 \cdot 2^{53} \approx 2.702 \cdot 10^{16}$

n	$\sigma(n)$
2	1
3	4
7	7
27	59
703	81
10 087	105
35 655	135
270 271	164
362 343	165
381 727	173
626 331	176
1 027 431	183
1 126 015	224
8 088 063	246
13 421 671	287
20 638 335	292
26 716 671	298
56 924 955	308
63 728 127	376
217 740 015	395
1 200 991 791	398
1 827 397 567	433
2 788 008 987	447
12 235 060 455	547
898 696 369 947	550
2 081 751 768 559	606
13 179 928 405 231	688
31 835 572 457 967	712
70 665 924 117 439	722
739 448 869 367 967	728
1 008 932 249 296 231	886

The computer search reported in this paper, which took approximately 4 CPU years, is still running. Please contact the author for the latest results.

4.1. **Theoretical analysis of the search speedup.** It is instructive to estimate the speedup of the program with respect to what we call a naive search. In a naive search, all initial values are tested, sequentially, to check if they are a t- or a σ-record-holder (or both). As explained before, it is sufficient to iterate T at n until $T^{(k)}(n) < n$. Thus, the number of iterates performed for each initial value is equal to its stopping time. The average stopping time is given by the formula

$$\sigma_{\text{naive}} = \sum_{k=1}^{\infty} k \, \frac{n_c(k)}{2^k}.$$

This number can be estimated using the data of Table 1, extended by the output of the recurrence formulas reported in the appendix. Performing the necessary computations gives the estimate

$$\sigma_{\text{naive}} \approx 3.493,$$

which agrees very well with empirical data. On the other hand, the average number of iterates required to test each initial value belonging to one of the congruence classes associated with the open nodes at depth 40 of the search tree is

$$\sigma_{40} = \frac{2^{40}}{n_{\text{o}}(40)} \sum_{k=41}^{\infty} (k-40)\, \frac{n_{\text{c}}(k)}{2^k}$$

(recall that the first 40 iterates are done almost for free), which gives the estimate

$$\sigma_{40} \approx 18.674.$$

This estimate also agrees very well with empirical data. The search speedup is thus

$$\frac{2^{40}}{n_{\text{o}}(40)}\, \frac{\sigma_{\text{naive}}}{\sigma_{40}} \approx 32.118.$$

It remains to analyze the speedup due to the final optimization discussed in section 3 (trajectory coalescence). Since $\gcd(2^k, 3^p) = 1$ for all $k, p \geq 0$, the integers belonging to one of the congruence classes modulo $2^k\,(3^p)$ are distributed uniformly among the congruence classes modulo $3^p\,(2^k)$. In particular, the 2^{10} initial values $n_{40}{\cdot}2^{40}{+}m_{40}$ that are tested for each type 3 node are distributed (almost) uniformly among the congruence classes modulo 9. We claim that the average number of iterates required to test the initial values belonging to one of these 9 congruence classes is the same for all 9 congruence classes. This follows from the observation that the numbers n_{40} which make n_0 belong to a given congruence class modulo 9 are uniformly distributed among the congruence classes modulo a power of 2. This forces the average number of iterates for each congruence class modulo 9 to be equal to that of any other congruence class (recall that the pruned tree involves the distribution of initial values among congruence classes modulo powers of 2). From this argument we conclude that the trajectory coalescence optimization is, in theory, completely independent of all previous optimizations, and so it introduces an extra speedup factor of 9/5 (only 5 of the 9 congruence classes modulo 9 need to be tested). This gives a total speedup factor of approximately 57.813.

The speedup factor just given disregards the extra time required to manage each one of the searches (the naive and ours), as well as the extra time required to identify the two types of record-holders. Since the average number of iterates required to test each initial value in the naive search is rather small, the (small) time required to test for t- and σ-record-holders may have a significant impact on the program speed. On the other hand, our search algorithm was designed to make these overheads small with respect to the time required to iterate $T(n)$ σ_{40} times (on the average). For example, the entire search tree can be generated in around one hour (on a DEC Alpha 266MHz machine), which is less than 0.1% of the total search time. The code that manages the two record-holder lists also contributes with a small fraction to the total search time. A direct comparison of the search times of the two algorithms gave a speedup of around 67, which shows that in our programs the overheads are considerably more significant in the naive search.

5. Concluding remarks

The main purpose of the search was to analyze the growth of the function $t(n)$. Figure 4 displays a graph with all t-record-holders, and their respective maximum excursions, found in the search. This figure suggests the following.

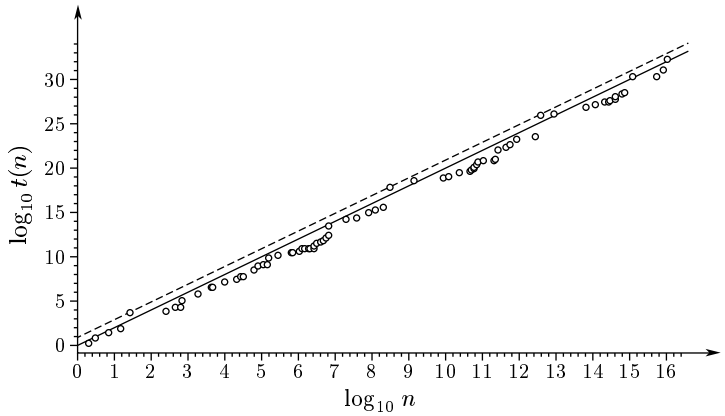

FIGURE 4. This log-log graph depicts all t-record-holders produced by the search reported in this paper. We also present the function $n \mapsto n^2$ (solid line) and the function $n \mapsto 8n^2$ (dashed line).

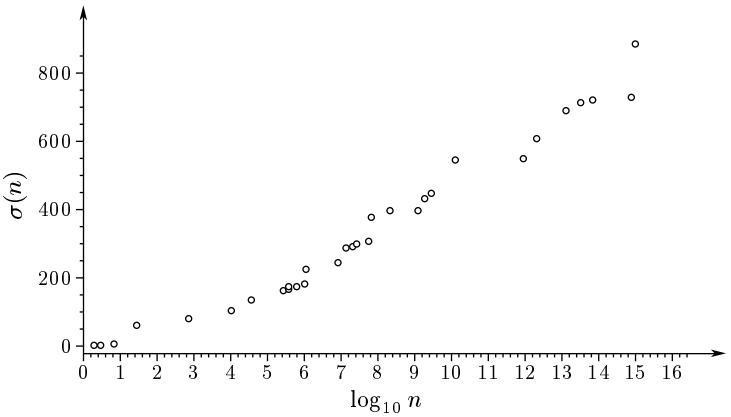

FIGURE 5. This log-linear graph depicts all σ-record-holders produced by the search reported in this paper.

Conjecture 1. *The maximum excursion $t(n)$ satisfies*

$$t(n) < n^2 \, f(n),$$

where $f(n)$ is either a constant or a very slowly increasing function of n.

With the data we have available it is not clear if $f(n)$ is, or isn't, a constant. In fact, $t(n) > n^2$ occurs for only seven t-record-holders for $n < 3 \cdot 2^{53}$, and in all cases found to date $t(n) < 8n^2$. This conjecture is in accord with theoretical results concerning the maximum excursions of a stochastic model of the $3x+1$ problem [2].

For completeness, Figure 5 displays a graph with all σ-record-holders found in the search, together with their respective stopping times. Unfortunately, the growth of the stopping times of σ-record-holders appears to be somewhat more erratic than the growth of the maximum excursion of t-record-holders. Nevertheless, Figure 5

shows that the stopping times of σ-record-holders appear to increase more or less linearly with the logarithm of the trajectory starting value. Given the average behavior of the stopping time, that was to be expected.

An important side result of our search is the verification of the $3x+1$ conjecture up to $n = 3 \cdot 2^{53} \approx 2.702 \cdot 10^{16}$. This upper limit can be used to improve the lower bound on the length of nontrivial cycles of $T(n)$ given in [1].

Acknowledgment

The author wishes to express his gratitude to J. C. Lagarias, for making available to the author his rather extensive annotated bibliography on the $3x + 1$ problem.

Appendix

In this appendix we present simple recurrence formulas to compute $n_{\mathrm{o}}(k)$ and $n_{\mathrm{c}}(k)$. It turns out that it is very convenient to break $n_{\mathrm{o}}(k)$ into smaller terms. Let $n_{\mathrm{o}}(k,p)$, $0 \le p \le k$, be the number of open nodes at depth k for which $y(k;m_k) = p$. Clearly, $n_{\mathrm{o}}(0,0) = 1$, and $n_{\mathrm{o}}(k) = \sum_{p=0}^{k} n_{\mathrm{o}}(k,p)$. When an open node at depth $k-1$, $k > 0$, is subdivided, one of the two nodes it generates will satisfy $y(k,m_k) = y(k-1,m_{k-1})$, and the other one will satisfy $y(k,m_k) = y(k-1,m_{k-1}) + 1$. If all the nodes generated by this process were open nodes, this would imply that $n_{\mathrm{o}}(k,p) = n_{\mathrm{o}}(k-1,p-1) + n_{\mathrm{o}}(k-1,p)$, with the assumption that $n_{\mathrm{o}}(k-1,-1) = n_{\mathrm{o}}(k-1,k) = 0$, giving a binomial distribution for $n_{\mathrm{o}}(k,p)$. However, some of the nodes generated may be closed nodes. Assuming that property 4 holds for all k, this happens when $3^p < 2^k$. Therefore we have, for $k > 0$,

$$n_{\mathrm{o}}(k,p) = \begin{cases} 0 & \text{if } 3^p < 2^k, \\ n_{\mathrm{o}}(k-1,p-1) + n_{\mathrm{o}}(k-1,p) & \text{if } 3^p > 2^k. \end{cases}$$

Defining $n_{\mathrm{c}}(k,p)$ in the same way as $n_{\mathrm{o}}(k,p)$ we also have

$$n_{\mathrm{c}}(k,p) = \begin{cases} n_{\mathrm{o}}(k-1,p) & \text{if } 3^p < 2^k, \\ 0 & \text{if } 3^p > 2^k. \end{cases}$$

(Actually, at most one of the numbers $n_{\mathrm{c}}(k,p)$ may be non-zero.) Knowing $n_{\mathrm{o}}(k,p)$ and $n_{\mathrm{c}}(k,p)$, it is a trivial task to compute $n_{\mathrm{o}}(k)$ and $n_{\mathrm{c}}(k)$. These recurrence formulas were verified with a computer for $k \le 40$.

References

[1] Shalom Eliahou, *The $3x+1$ problem: new lower bounds on nontrivial cycle lengths*, Discrete Mathematics **118** (1993), no. 1–3, 45–56. MR **94h:**11017

[2] J. C. Lagarias and A. Weiss, *The $3x+1$ problem: two stochastic models*, The Annals of Applied Probability **2** (1992), no. 1, 229–261. MR **92k:**60159

[3] Jeffrey C. Lagarias, *The $3x+1$ problem and its generalizations*, The American Mathematical Monthly **92** (1985), no. 1, 3–23. MR **86i:**11043

[4] G. Leavens and M. Vermeulen, *$3x+1$ search programs*, Computers and Mathematics, with Applications **24** (1992), no. 11, 79–99. MR **93k:**68047

Departamento de Electrónica e Telecomunicações / INESC Aveiro, Universidade de Aveiro, 3810 Aveiro, Portugal

E-mail address: tos@inesca.pt