On Two Mechanisms Related to the “3n+1” Problem

Brenton Bostick
Wittenberg University
Post Office Box 720
Springfield, Ohio 45501-0720
s04.bbostick@wittenberg.edu

ABSTRACT

Let T(n) = (3n+1)/2 if n odd, and n/2 if n even. The “3n+1” problem asks if for all integers $n \geq 1$, there exists an i such that $T^i(n) = 1$. In this paper, I present two mechanisms related to the problem. The first is a cellular automaton that computes iterations of the “3n+1” function. The second is a function that computes successive local maxima and minima of the sequence generated by iterating T.

1. INTRODUCTION

Let T(n) = (3n+1)/2 if n odd, and n/2 if n even. The “3n+1” problem, known also as the Collatz problem, asks if for all integers $n \geq 1$, there exists an i such that $T^i(n) = 1$. An excellent survey of progress on the problem has been written by Lagarias [3]. Wolfram has many interesting results on computational aspects of the problem [5].

2. A CELLULAR AUTOMATON FOR COMPUTING THE FUNCTION T

A cellular automaton is an array of cells on a grid that evolves through discrete time steps. Each cell has a state that is updated according to previously defined rules that depend on the states of the cell’s neighbors and the state of the cell itself. Cellular automata are simply defined yet demonstrate behavior that is sometimes chaotic and unpredictable.

The function T(n) can be restated as (n+1)(n mod 2) + $\lfloor n/2 \rfloor$. This particular arrangement makes T(n) amenable for implementation on a computer. With the bits of n expressed as $\ldots d_3d_2d_1d_0$, n mod 2 is simply d_0 and $\lfloor n/2 \rfloor$ is $\ldots d_4d_3d_2d_1$. Arranged as a traditional addition problem (with π acting as the binary point):

$$\begin{array}{lcccccc} & \ldots & d_4\,d_0 & d_3\,d_0 & d_2\,d_0 & d_1\,d_0 & d_0\,d_0 & \pi \\ & \ldots & d_5 & d_4 & d_3 & d_2 & d_1 & \pi \\ + & & & & & & d_0 & \pi \\ \hline & \multicolumn{6}{c}{T\ (\ \ldots\ d_5\,d_4\,d_3\,d_2\,d_1\,d_0)} \end{array}$$

Figure 1

In the process of the addition, we keep track of the running sum, carry bit and the d_0 bit. The trick is that multiple iterations can be computed in parallel.

Starting at the right, we keep track of the intermediate sum $d_1 + 2d_0$. To symbolize this in the cellular automaton, we say that cell j in the neighborhood (i, j, π), where j is a 0 or 1, goes to s(i+2j , j), as shown in Figure 2. The notation s(a,b) represents a state that encodes the intermediate sum a and the d_0 bit value of b. All other states remain the same.

$$\begin{array}{cccccc} \ldots & d_2 & d_1 & d_0 & \pi \\ \ldots & d_2 & d_1 & s\,(\nwarrow + 2\uparrow,\ \uparrow) & \pi \end{array}$$

Figure 2

After the second time step, the previous intermediate sum is reduced modulo 2 and the next intermediate sum is computed.

$$\begin{array}{ccccc} \ldots & d_2 & d_1 & d_0 & \pi \\ \ldots & d_2 & d_1 & s\,(\nwarrow + 2\uparrow,\ \uparrow) & \pi \\ \ldots & d_2 & s\,(m\,(\nwarrow) + \uparrow z\,(\nearrow) + q\,(\nearrow),\ z\,(\nearrow)) & m\,(\uparrow) & \pi \end{array}$$

Figure 3

The function m(n) is m mod 2, the function q(n) is the quotient of a divided by 2 if n = s(a,b), and the function z(n) returns b if n = s(a,b). The process continues for the specified number of time steps. We can assign the 6 possible intermediate sums and d_0 combinations the following integers:

s(0,0)	→ 2
s(1,0)	→ 3
s(0,1)	→ 4
s(1,1)	→ 5
s(2,1)	→ 6
s(3,1)	→ 7

It is easy to define m, q, and z in terms of numerical functions:

$$q(n) = \lfloor (n-2)/4 \rfloor$$
$$m(n) = n \bmod 2$$
$$z(n) = \lfloor n/4 \rfloor$$
$$s(a,b) = 2 + a + 2b.$$

For consistency in using integers for states, we will use 8 instead of π for the binary point. The update rules for a cell j in a neighborhood (i,j,k) are:

```
if ($j$ = 0 or 1) then
{
        if ($k$ = 8) then
                update j to $2 + m(i) + 4j$
        else if ($k$ = 2,3,4,5,6, or 7) then
                update j to $2 + m(j) + q(k) + (2 + j) z(k)$
}
else if ($j$ = 2,3,4,5,6, or 7) then
        update j to $m(j)$
else leave j alone
```

Below is an example run of the cellular automaton with input $27 = 11011_2$ after 10 time steps:

```
0 0 0 0 0 1 1 0 1 1 8
0 0 0 0 0 1 1 0 1 7 8
0 0 0 0 0 1 1 0 6 1 8
0 0 0 0 0 1 1 6 0 6 8
0 0 0 0 0 1 7 0 5 0 8
0 0 0 0 0 6 1 5 1 3 8
0 0 0 0 5 0 5 1 3 1 8
0 0 0 4 1 5 1 3 1 7 8
0 0 4 0 5 1 3 1 7 1 8
0 4 0 4 1 3 1 7 1 7 8
4 0 4 0 2 1 7 1 7 1 8
```

Figure 4

The cellular automaton evolves downward and each cell reflects the update of the cell directly above it. On a larger time scale, labeling each cell with a number would be infeasible, so each cell is colored uniquely to demonstrate the same effect. Below are 30 time steps of the cellular automaton with an input of $27 = 11011_2$:

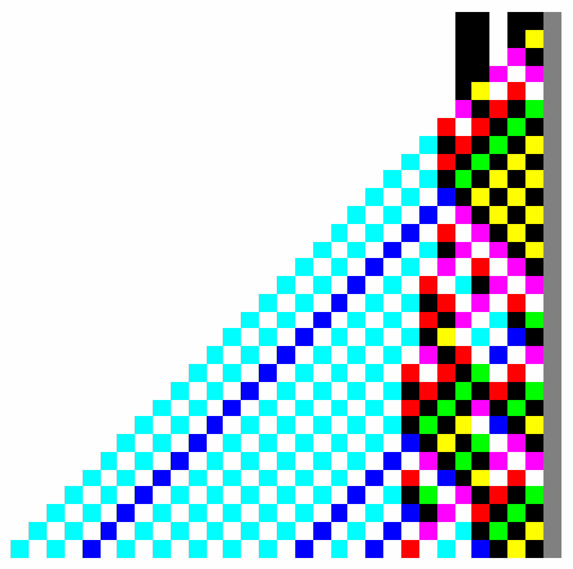

Figure 5

Additional rules can be included to remove the 2-colored trails that propagate to the left. These cells have states of 2 and 4, which indicate that both have a sum value of 0 and a d_0 bit value of 0 and 1, respectively.

```
0 0 1 1 0 1 1 8
0 0 1 1 0 1 7 8
0 0 1 1 0 6 1 8
0 0 1 1 6 0 6 8
0 0 1 7 0 5 0 8
0 0 6 1 5 1 3 8
0 5 0 5 1 3 1 8
4 1 5 1 3 1 7 8
0 5 1 3 1 7 1 8
4 1 3 1 7 1 7 8
0 2 1 7 1 7 1 8
```

Figure 6

The maximum number of cells required with the new rule now depends on the length of the largest integer in the sequence of iterates of T.

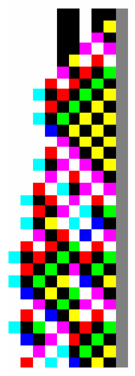

Figure 7

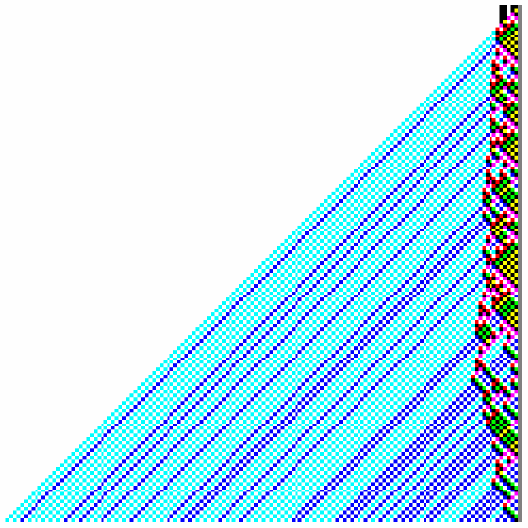

Figure 8

The cellular automaton needs 2 time steps to compute each bit, but multiple bits are computed in parallel. After an initial n time steps for an input n bits long, the border of active cells grows as the actual sequence of iterations of t. If only the sequence of even/odd decisions is needed, then for n time steps, only the first $\lfloor n/2 \rfloor$ bits of the input are required.

Figure 8 shows the evolution of the cellular automaton from input 27. The function T reaches 1 in 70 steps, so the cellular automaton reaches the (218) state in 140.

The cellular automaton that has been defined is simply a codification of the process of computing T by addition.

The "3n+1" problem can now be restated by asking if the cellular automaton does indeed always reach the (218) state for all valid inputs.

3. A "FASTER" FUNCTION

A function U is introduced that computes the local maxima and minima of the sequence generated by iterating T.

A run is a group of consecutive equal bits. A single bit can be a run. Define a function r(n) such that r(n) returns the length of the run of trailing bits in the binary representation of n. Example: $r(10110_2) = 1$ and $r(1011_2) = 2$. Define a function a(n) that returns the integer whose binary representation is the bits that occur after the run of trailing bits in n. Example: $a(1011_2) = 10_2 = 2$ and $a(10110_2) = 1011_2 = 11$. For integers of the form $n = 2^k-1$, a(n) is defined to be 0.

For even n, T(n) is n/2 and r(n) is the number of trailing 0's in the binary representation of n.

For odd n, T(n) is equal to (n + 2n + 1)/2 and r(n) is the number of trailing 1's in the binary representation of n. Necessarily, the last bit of a(n) for any odd n is 0.

Define a function U on the integers such that $U(n) = T^{r(n)}(n)$. Then U(n) = a(n) if n is even and $3^{r(n)}(a(n)+1)-1$ if n is odd.

If n is even, then U(n) simply shifts out all of the factors of 2 and is a(n). The odd branch of U(n) is derived from the fact that if an odd n has r(n) many trailing 1's, then T(n) has r(n)-1 trailing 1's. The number of odd iterations of an odd n is r(n), as can be shown by induction. The actual expression in the function is based on $(3/2)^k(n+1)-1$, which is the expression for iterating T(n) k many times, for odd n. The factor of $1/2^k$ may be removed because the propagation of carry bits implies $(n+1)/2^k = a(n) + 1$, for odd n.

The function U computes successive maxima and minima of T because a run of evenness implies a minimum and a run of oddness implies a maximum.

Lemma The mean number of steps of T skipped by U is 2.

Proof: The function r defined above represents the number of "trials" before "failure" of the run of bits of n. Because the probability of success is the same for each trial (1/2) and each trial is independent of each other, r is a geometric random variable with p = 1/2. The function r has a geometric distribution over the integers. A geometric random variable has a mean of 1/p. Therefore, the mean value of r(n) is 2.

For any n, the number of steps of T skipped by U is r(n). Since the mean value of r is 2, the mean number of steps skipped by T is 2.

If an integer converges to 1 under iteration of T, then it will on average take half as many steps to do so under U. If an integer does not converge to 1, then nothing more can be said.

4. SUMMARY

The mechanisms shown in this article are new tools in the analysis of the "3n+1" problem. The cellular automaton demonstrates what effect that locality has on the bits on computing the function T. The function U demonstrates the symmetry between evenness and oddness in the problem, and the amount of latent information in the integers themselves.

5. ACKNOWLEDGEMENTS

My thanks go to the participants of the NKS Summer School 2003 for providing the environment in which these ideas were first cultivated, Matthew Szudzik for his support on this project, and Dr. Doug Andrews for helping me clarify portions of my arguments.

6. References

[1] Conway, J.H. Unpredictable Iterations. Proc. 1972 Number Th. Conf., University of Colorado, Boulder, Colorado, 49-52. 1972.

[2] Knuth, D.E. The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, 3rd ed. Reading, MA: Addison-Wesley, 1998.

[3] Lagarias, J.C. The 3x+1 Problem and Its Generalizations. Amer. Math. Monthly. 92, 3-23. 1985.

[4] Terras, R. A Stopping Time Problem on the Positive Integers. Acta. Arith. 30, 241-252, 1976.

[5] Wolfram S. A New Kind Of Science. Champaign, IL: Wolfram Media, 2002