2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovations

Collatz Conjecture for $2^{100000} - 1$ is True - Algorithms for Verifying Extremely Large Numbers

Wei Ren
School of Computer Science, Hubei Key Laboratory of Intelligent Geo-Information Processing, China University of Geosciences,
Wuhan, P. R. China, 430074
Guizhou Provincial Key Laboratory of Public Big Data, Guizhou University, Guizhou, P. R. China
weirencs@cug.edu.cn

Simin Li
School of Computer Science, China University of Geosciences,
Wuhan, P. R. China, 430074
446427812@qq.com

Ruiyang Xiao
School of Computer Science, School of Mathematics and Physics, China University of Geosciences,
Wuhan, P. R. China, 430074
ruiyangxiaomath@cug.edu.cn

Wei Bi
SeeleTech Corporation, San Francisco, USA, 94107
wei.bi@alumni-oxford.com

***Abstract*—Collatz conjecture (or 3x+1 problem) has been explored for about 80 years. By now the largest number that has been verified for Collatz conjecture is about 60 bits. In this paper, we propose new algorithms that can verify much greater numbers than known algorithms, i.e., about 100000 bits (30000 digits) on whether they can return 1 after 3*x+1 and x/2 computations. The proposed algorithms change numerical computation to bit computation (or logical computation), and change inner memory data representation to hard disk file manipulation. We make it possible to verify extremely large numbers without concerning computer memory upper-bound. The largest number verified for Collatz conjecture until now in the world is $2^{100000} - 1$, and this magnitude of extremely large numbers has never been verified. We discovered that this number can return to 1 after 481603 times of 3*x+1 computation, and 863323 times of x/2 computation. At last, we stress that some open problems may be used for the green mining in blockchain-based cryptocurrencies, e.g., BTC. We discuss the applicability of Collatz conjecture for the Proof of Work in Blockchain.**

***Index Terms*—Collatz Conjecture; 3x+1 Problem; Computational Algebra; Algorithmic Number Theory; Numerical Algorithm**

I. Introduction

The Collatz conjecture is a mathematical conjecture that is first proposed by Lothar Collatz in 1937. It is also known as the 3x+1 conjecture, the Ulam conjecture, the Kakutani's problem, the Thwaites conjecture, or the Syracuse problem [2].

Simply speaking, the conjecture can be stated as follows. Take any positive integer number x. If x is even, divide it by 2 to get $x/2$. If x is odd, multiply it by 3 and add 1 to get $3 * x + 1$. Repeat the process again and again. The Collatz conjecture is that no matter what the number (i.e., x) is taken, the process will always eventually reach 1.

The contributions of the paper are listed as follows:

1) A new computing algorithm is proposed that can verify Collatz conjecture up to a new upper bound. The proposed algorithm changes numerical computation to logical computation, and compute $3 * x + 1$ bit by bit in binary. This upper bound is significantly larger than any existing known algorithms and experimental witness. The current known upper bound for starting value that has been checked (or can be checked) is up to about 60bits [1], [2], but the proposed algorithm in this paper can check starting numbers up to 100000bits.
2) A new computing algorithm is proposed that can verify Collatz conjecture up to theoretically unlimited upper bound, only depending on the timing cost. The rationale in the algorithm is using outer memory (such as a data file in a hard disk) instead of inner memory (such as an array or allocated memory) for data processing.

The rest of the paper is organized as follows. Section II presents relevant background. Section III details our proposed algorithms and analysis. Section IV provides some experimental results. Finally, Section V concludes the paper.

II. Preliminary

Definition Collatz Transformation

$$CT(x) = \begin{cases} TPO(x) = 3 * x + 1 & (TPO) \quad (x \in [1]_2), \\ H(x) = x/2 & (H) \quad (x \in [0]_2), \end{cases} \tag{1}$$

The research was financially supported by Major Scientific and Technological Special Project of Guizhou Province (20183001), the Open Funding of Guizhou Provincial Key Laboratory of Public Big Data (2017BDKFJJ006), and Open Funding of Hubei Provincial Key Laboratory of Intelligent Geo-Information Processing (KLIGIP2016A05).

978-1-5386-9380-3/18/$31.00 ©2018 IEEE
DOI 10.1109/SmartWorld.2018.00099

where $[1]_2 = \{a|a \equiv 1 \mod 2, a \in \mathbb{N}^*\}$; $[0]_2 = \{a|a \equiv 0 \mod 2, a \in \mathbb{N}^*\}$; $\mathbb{N}^* = \{a|a \in \mathbb{Z}, a \geq 1\}$. $TPO(x)$ can be simply denoted as TPO, and $H(x)$ can be simply denoted as H.

Definition The Collatz Conjecture. $\forall x \in \mathbb{N}$, after finite times of Collatz Transformation $x \Leftarrow CT(x)$, x will become 1. (Here "$\Leftarrow$" is assignment symbol and "$x \Leftarrow y$" means to assign value y to x.)

For example, dynamics after each Collatz Transformation from a starting value to 1 are as follows:

1) $1 \to 4 \to 2 \to 1$;
2) $7 \to 22 \to 11 \to 34 \to 17 \to 52 \to 26 \to 13 \to 40 \to 20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$.

III. Proposed Algorithms

Two notations will be used in the following presentations as follows:

1) (x) returns the least significant bit (LSB) of binary number x.
2) $\langle x \rangle$ returns the most significant bit (MSB) of binary number x.

Simply speaking, if $x \leq 3$, (x) returns rightmost bit of binary representation of x, and $\langle x \rangle$ returns leftmost bit of binary representation of x.

For example, $(10) = 0$, $\langle 10 \rangle = 1$, where 10 is a binary number. Besides, $(1{+}0) = 1, (1{+}1) = 0, \langle 1{+}1 \rangle = 1, \langle 1{+}0 \rangle = 0$.

Fig.1 depicts the design rationale in the proposed algorithm. Suppose the length of binary representation on x is n. $a[0]$ is MSB (leftmost bit) of x, and $a[n-1]$ is LSB (rightmost bit) of x. $2{*}x$ is one bit of left shift. The LSB of $2{*}x$ is thus 0. $3{*}x{+}1$ can be looked as $x + 2x + 1$. Suppose the resulting summation is $b[0]b[1]...b[n-1]$. The carrier bit is $c[0]c[1]...c[n-1]$. As x is odd (due to $3 * x + 1$ is to be computed), $a[n-1] = 1$. The rightmost bit of result (i.e., $3 * x + 1$) is $(a[n-1] + 0 + 1)$, which is $(1 + 0 + 1) = 0$. The carrier bit at this location is $c[n-1] = \langle a[n-1] + 0 + 1 \rangle = \langle 1 + 0 + 1 \rangle = 1$. Next, $b[n-2] = (a[n-2] + a[n-1] + c[n-1])$, where $a[n-1] = 1$ and $c[n-1] = 1$. Similarly, further computation for other bits follows $b[n-k] = (a[n-k+1] + a[n-k] + c[n-k+1])$, $c[n-k] = \langle a[n-k+1] + a[n-k] + c[n-k+1] \rangle$.

Major computational logics in the proposed algorithm are listed in Eq.3 and Eq.2. Note that, before the computation of $3 * x + 1$, x is firstly represented as binary string like $a[0]\|a[1]\|...\|a[n-2]\|a[n-1]$ from MSB to LSB. The computation is from LSB to MSB.

$Head$ is the font one bit or two bits of $3{*}x{+}1$, depending on $c[0] = 0$ or not. $\underline{1}$ is one bit; $\underline{10}$ are two bits. The final result of $3 * x + 1$ is represented as binary string like $Head\|b[0]\|b[1]\|...\|b[n-2]\|b[n-1]$, where $Head$ is one bit "1" or two bits "10". $\|$ is concatenation.

$$Head = \begin{cases} c[0] + a[0] = c[0] + 1 = 0 + 1 = \underline{1}, & c[0] = 0, \\ c[0] + a[0] = c[0] + 1 = 1 + 1 = \underline{10} & c[0] = 1. \end{cases} \quad (2)$$

After above preparations (Fig.1, Eq.3 and Eq.2), Algorithm 1 is proposed as follows:

Data: x
Result: $TPO(x)$. That is, $result = TPO(x) = 3 * x + 1$.
$b[n-1] \Leftarrow' 0'$;
$c[n-1] \Leftarrow 1$;
for $(i = n-2; i >= 0; i--)$ **do**
  $sum \Leftarrow a[i+1] + a[i] + c[i+1]$;
  if $sum == 2 || sum == 3$ **then**
    $c[i] \Leftarrow 1$;
  end
  if $sum == 0 || sum == 1$ **then**
    $c[i] \Leftarrow 0$;
  end
  if $sum == 0 || sum == 2$ **then**
    $b[i] \Leftarrow' 0'$;
  end
  if $sum == 1 || sum == 3$ **then**
    $b[i] \Leftarrow' 1'$;
  end
end
if $c[0] == 1$ **then**
  $result \Leftarrow'' 10'' \| result$;
end
if $c[0] == 0$ **then**
  $result \Leftarrow' 1' \| result$;
end
return $result$;

Algorithm 1: Input an extremely large number x that is represented in binary. Output $TPO(x)$ that is $3 * x + 1$.

Remark Algorithm 1 changes numerical computation of $3 * x + 1$ into simple bit computation. That is, only addition of three bits is conducted (in $sum \Leftarrow a[i+1] + a[i] + c[i+1]$), avoiding to compute $3{*}x$ directly in which x is upper-bounded by memory limitation in computer. Thus, it becomes possible to compute $3 * x + 1$ of an extremely large number by this algorithm.

Furthermore, the numerical computation $sum \Leftarrow a[i+1] + a[i] + c[i+1]$ can be changed to logical computation as follows (Algorithm 2):

Remark Algorithm 2 computes a starting number x with binary length n in $O(n)$ time. Besides, in each time of loop execution (i.e., For), only one decision instruction (i.e., If) is executed. The result (i.e., $3 * x + 1$) can be computed in one pass traverse of x from LSB (i.e., $a[n-1]$) to MSB (i.e., $a[0]$).

Above two algorithms for computing $TPO(x)$ are used for outputting and analyzing the dynamics of Collatz transformations on extremely large inputting numbers. That is, an algorithm outputting an entire procedure from a extremely large starting number to 1 by invoking $TPO(x)$ is given as follows (Algorithm 3):

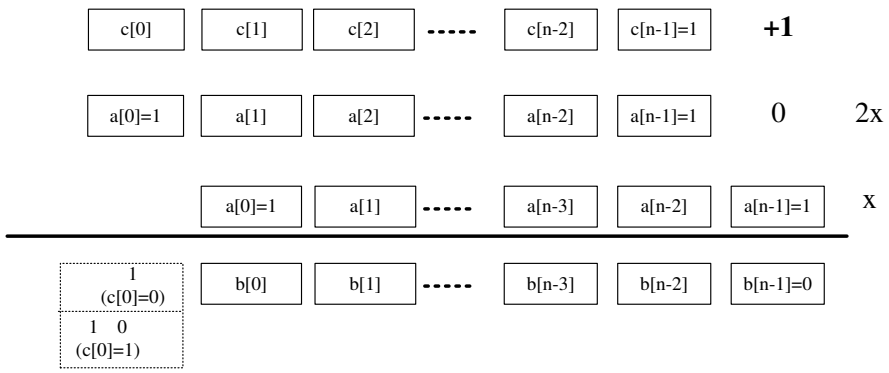

Fig. 1. Fast algorithm for $3 * x + 1$ (TPO) on an extremely large number x via bits computation instead of numerical computation.

$$\begin{cases} b[n-1] = (a[n-1] + 0 + 1) = (1 + 0 + 1) = 0 \\ c[n-1] = \langle a[n-1] + 0 + 1 \rangle = \langle 1 + 0 + 1 \rangle = 1, \\ b[n-2] = (a[n-1] + a[n-2] + c[n-1]) = (a[n-1] + a[n-2] + 1) \\ c[n-2] = \langle a[n-1] + a[n-2] + c[n-1] \rangle = \langle a[n-1] + a[n-2] + 1 \rangle, \\ \ldots \\ b[n-k] = (a[n-k+1] + a[n-k] + c[n-k+1]) \\ c[n-k] = \langle a[n-k+1] + a[n-k] + c[n-k+1] \rangle, \\ \ldots \\ b[1] = (a[1] + a[2] + c[2]) \\ c[1] = \langle a[1] + a[2] + c[2] \rangle, \\ b[0] = (a[0] + a[1] + c[1]) = (1 + a[1] + c[1]) \\ c[0] = \langle a[0] + a[1] + c[1] \rangle = \langle 1 + a[1] + c[1] \rangle. \end{cases} \tag{3}$$

The procedure (or dynamics) from a starting number to 1 consists of two Collatz transformations or computations, which is $3 * x + 1 (TPO)$ and $x/2 (H)$. The dynamics are outputted by Algorithm 3 in terms of a serial of Collatz transformations. The dynamics is represented by $code$ consisting of "$-$" and "0", which denotes $H(TPO(x)) = (3 * x + 1)/2$ and $x/2$, respectively. U is the occurred times of TPO, and D is the occurred times of H in dynamics (or $code$).

Proposition 3.1: If TPO occurs, H occurs intermediately after it. That is, "$TPO,\ H$" always together occurs in any dynamics.

Proof If $x \in [1]_2$, $TPO(x) = 3 * x + 1 \in [0]_2$. Thus, the next Collatz transformation immediately after "TPO" must be "H". Therefore, H always occurs after a TPO transformation. That is, "TPO" and H always *together* occurs. ∎

Therefore, $(3 * x + 1)/2$ can be written together as $H(TPO(\cdot)$, and denoted as "$-$" in outputting code in our computer program. For better outputting vision, we denote $H(x)$ as "0" in outputting code in our program.

Example The outputting code (or dynamics) for $x = 5$ is "-000", which means $H(TPO(\cdot))$, $H(\cdot)$, $H(\cdot)$, $H(\cdot)$.

In other words, $5 \to 16 \to 8 \to 4 \to 2 \to 1$.

U in $code$ for 5 is 1, because there are 1 "$-$" in $code$. D in $code$ for 5 is 4, because there are 3 "0" in $code$ and 1 "$-$" in $code$.

Besides, to support extremely large numbers that is even in binary, we need to break the limitation of inner memory in computer (e.g., the array size, the size of data type such as "long long integer"). Therefore, in the implementation of Algorithm 3, to break the limitation of inner memory, we use a data file in hard disk to process those extremely large numbers (see Algorithm 4 in Appendix). That is, the reading and writing operation are changed to "from a data file in hard disk", instead of "from an data array or an allocated memory in inner memory". Thus, it can support any length of manipulated number without any upper bound (i.e., starting number x and occurred intermediate numbers during the procedure can be unlimited large).

IV. Experiment Results

Example Some examples for dynamics from starting number x to final number 1 are listed as follows (recall that $-$ represents $H(TPO(x))$, or $(3 * x + 1)/2$; 0 represents $H(x)$, or $x/2$):

Data: x
Result: $TPO(x)$. That is, $result = TPO(x) = 3 * x + 1$.
$b[n-1] \Leftarrow' 0'$;
for $(i = n-2; i >= 0; i--)$ **do**
  if $(a[i+1] == 1)\&\&(a[i] == 1)\&\&(c[i+1] == 1)$ **then**
    $c[i] \Leftarrow 1$;
    $b[i] \Leftarrow' 1'$;
  end
  if $(a[i+1] == 1)\&\&(a[i] == 0)\&\&(c[i+1] = 1)$ **then**
    $c[i] \Leftarrow 1$;
    $b[i] \Leftarrow' 0'$;
  end
  if $(a[i+1] == 0)\&\&(a[i] == 1)\&\&(c[i+1] = 1)$ **then**
    $c[i] \Leftarrow 1$;
    $b[i] \Leftarrow' 0'$;
  end
  if $(a[i+1] == 1)\&\&(a[i] == 1)\&\&(c[i+1] == 0)$ **then**
    $c[i] \Leftarrow 1$;
    $b[i] \Leftarrow' 0'$;
  end
  if $(a[i+1] == 1)\&\&(a[i] == 0)\&\&(c[i+1] == 0)$ **then**
    $c[i] \Leftarrow 0$;
    $b[i] \Leftarrow' 1'$;
  end
  if $(a[i+1] == 0)\&\&(a[i] == 1)\&\&(c[i+1] == 0)$ **then**
    $c[i] \Leftarrow 0$;
    $b[i] \Leftarrow' 1'$;
  end
  if $(a[i+1] == 0)\&\&(a[i] == 0)\&\&(c[i+1] == 1)$ **then**
    $c[i] \Leftarrow 0$;
    $b[i] \Leftarrow' 1'$;
  end
  if $(a[i+1] == 0)\&\&(a[i] == 0)\&\&(c[i+1] == 0)$ **then**
    $c[i] \Leftarrow 0$;
    $b[i] \Leftarrow' 0'$;
  end
end
if $c[0] == 1$ **then**
  $result \Leftarrow'' 10''\|result$;
end
if $c[0] == 0$ **then**
  $result \Leftarrow' 1'\|result$;
end
return $result$;

Algorithm 2: Input an extremely large number x that is represented in binary. Output $TPO(x)$ that is $3 * x + 1$.

Data: x
Result: $code, U, D-U, ratio$
while $current \neq 1$ **do**
  if $IsEven(current) \neq 1$ **then**
    $result \Leftarrow TPO(current)$;
    % It is the intermediate one H after TPO ;
    $flag\_first\_down \Leftarrow 1$;
    $U \Leftarrow U + 1$;
    $continue$;
  end
  else
    if $flag\_first\_down == 1$ **then**
      $code \Leftarrow code\|-$;
      $D \Leftarrow D + 1$;
      $flag\_first\_down \Leftarrow 0$;
    end
    else
      $code \Leftarrow code\|0$;
      $D \Leftarrow D + 1$;
    end
    $continue$;
  end
end
return $code, U, D-U, ratio = (D-U)/U$;

Algorithm 3: Input an extremely large number x. Output $code$ that presents the dynamics of x, U, $D-U$, and $ratio = (D-U)/U$.

1) $x = 7$, $(x)_2 = 111$. $(x)_2$ means x is represented in binary.
   Dynamics (intermediate numbers after Collatz transformations):
   $1011 \to 10001 \to 11010 \to 1101 \to 10100 \to 1010 \to 101 \to 1000 \to 100 \to 10 \to 1$.
   Thus, code is "$---0-00-000$". $U = 5$, $D-U = 6$, $ratio = (D-U)/U = 1.2000000$.
2) $x = 63$, $(x)_2 = 111111$.
   Code is "$------000--0---0----0--00---0--0------00----000-0-0-000-00---0000-000$".
   $U = 39$ $D-U = 29$ $ratio = (D-U)/U = 0.7435898$.

Example W.l.o.g., we select numbers with binary form liking 1^{MAXLEN} as testing examples. E.g., $1^{100} = \underbrace{111...11}_{100}$, $1^{1000} = \underbrace{111...11}_{1000}$, and so on.

The dynamics of the largest number x that has been computed until now is obtained. $(x)_2 = 1^{100000} = \underbrace{111...1}_{100000}$. In its dynamics, the total times of $3 * x + 1$ (i.e., U) is 481603; the total times of $x/2$ (i.e., H) is $381720+481603 = 863323$. $ratio = (D-U)/U = 381720/481603 = 0.7926030$. In other words, this number endures 481603 times of TPO Collatz transformation, and 863323 times of H Collatz transformations to become 1.

The details of data is provided as supplement (upon request by email or on my personal web site), as it is too long to present in the paper.

Some other examples are given in Tab. I.

Discussion Currently, blockchain-based cryptocurrenccies (e.g., BTC) usually use hash functions for the Proof of Work (PoW), which cost or waste a large amount of electrical power. We argue that some open problems in number theory may be used for "green mining" by replacing hash computation, as hash computation is meaningless in terms of creating knowledge. We need to find some functions to replace hash function, which can only be computed by random trial but is meaningful in terms of producing some knowledge. Those functions may be selected from certain open problems in number theory.

Because we have already verified the largest number in the world, namely, $2^{100000}-1$, that can guarantee Collatz conjecture. We may continue to believe that Collatz conjecture is true. If we can find a counterexample during PoW computation, the computation of PoW will become a significant finding. The design rationale for PoW is as follows:

Give x and n, miners in blockchain ecosystems try to find y such that $f(CT(x||y)) > r$, where $CT(\cdot)$ is Collatz Transformation; r is a requirement, for the ratio computed from the dynamics of $x||y$; $f(a)$ returns $ratio = (D-U)/U$, where U is the number of 0 ($3*x+1$ or $HPO(\cdot)$) and D is the number of 1 ($x/2$ or $H(\cdot)$) in the dynamics of inputting binary number $a \in \{0,1\}^{\geq 1}$. Recall that $CT(\cdot) \in \{0,1\}^{\geq 1}$. If it can be accomplished only by trials of y, it will be fine. For the better understanding, we explain the design by comparing it with PoW in BTC as follows: r corresponds to the number of 0 in the head of hash value; $f(CT(\cdot))$ corresponds to the hash function; x corresponds to the concatenation of values such as previous block hash, timestamp, difficulty, and so on; y corresponds to $nonce$.

V. Conclusion

In this paper, we proposed several algorithms that can be used for verifying Collatz conjecture upto extremely large numbers. The advantages of the proposed algorithms are as follows:

1) The computation is only symbol logical computation (i.e., $\wedge$, $\vee$), which is very fast comparing with numerical computation.
2) The computation of $3*x+1$ for extremely large number x becomes possible, as x is represented in binary and computed bit by bit.
3) The computation has no memory limitation. Theoretically, $x = 2^{+\infty}$, as intermediate and final computation results are written to a file in hard disk instead of to an array or allocated memory in inner memory. The experimental results show that the largest number $2^{100000}-1$ has been verified until now (i.e., with binary form $\underbrace{111...11}_{100000}$), can return to 1, after 481603 times of $3*x+1$ transformation and 863323 times of $x/2$ transformation.

References

[1] Silva, Tomas Oliveira e Silva. Computational verification of the 3x+1 conjecture. *http://sweet.ua.pt/tos/3x+1.html*, Retrieved 28 Jan. 2016.

[2] Wikipedia. Collatz conjecture. *https://en.wikipedia.org/wiki/Collatz_conjecture*, Retrieved 28 Jan. 2016.

Appendix

Algorithm 4.

Data: The file with the file name "$TESTNUMBER$"
Result: The file with the file name "$CODE$"
$GenTestNumber(''TESTNUMBER'');$
$FileCopy(\text{“}TESTNUMBER'',''input\_start'');$
$FileCopy(\text{“}TESTNUMBER'',''current\_x'');$
while $(GetFileLength(\text{”}current\_x\text{”})! = 1)$ **do**
  if $IsEven(''current\_x'') == 0$ **then**
    $TPO(''current\_x'',''temp'');$
    $FileCopy(''temp'',''current\_x'');$
    $Firstdown \Leftarrow 1;$
    $U \Leftarrow U + 1;$
    $continue;$
  end
  else
    if $Firstdown == 1$ **then**
      $CutFileRear(''current\_x'');$
      $AppendToFile(''CODE'',' -');$
      $D \Leftarrow D + 1;$
      $Firstdown \Leftarrow 0;$
      $FileAppend(''current\_x'',''DYNAMICS'');$
    end
    else
      $CutFileRear(''current\_x'');$
      $AppendToFile(''CODE'',' 0');$
      $D \Leftarrow D + 1;$
      $FileAppend(''current\_x'',''DYNAMICS'');$
    end
    $continue;$
  end
end
$ratio \Leftarrow (D-U)/U;$
return $U, D, ratio;$

Algorithm 4: Main algorithm for outputting dynamics. Output $code$ of x to a file named "$CODE$", where x is the starting number that is the content of the file named "$TESTNUMBER$".

Example Some examples for dynamics from starting number x to final number 1 are listed as follows:

1) $x = 2^{100} - 1$. $(x)_2 = 1^{100} = \underbrace{111...11}_{100}$.

TABLE I

x IS IN BINARY WITH THE FORM 1^{MAXLEN}, AND ITS LENGTH IS $MAXLEN$. THE DYNAMICS IS RECORDED FROM STARTING NUMBER TO 1.

$MAXLEN$	x in binary	digits of x	$(U, D-U)$	$ratio$
100	$1^{100}=\underbrace{111...1}_{100}$	30	(528, 409)	0.7746212
500	$1^{500}=\underbrace{111...1}_{500}$	150	(2417, 1914)	0.7918908
1000	$1^{1000}=\underbrace{111...1}_{1000}$	300	(4316, 3525)	0.8167285
5000	$1^{5000}=\underbrace{111...1}_{5000}$	1500	(24131, 19116)	0.7921761
10000	$1^{10000}=\underbrace{111...1}_{10000}$	3000	(48126, 38152)	0.7927524
50000	$1^{50000}=\underbrace{111...1}_{50000}$	15000	(239020, 189818)	0.7941511
100000	$1^{100000}=\underbrace{111...1}_{100000}$	30000	(481603, 381720)	0.7926030

$$
\begin{aligned}
&----------0000-00---00-\\
&000--0--000-000--0000-0--00---00-0-\\
&--00-0---0---0-00---0-0---0000-\\
&-00-0-00---0-0000---00------0-00-\\
&0000-000--000-000---0-00-00-0--00---\\
&---0------00---0-0-0000-0--000-0-\\
&---0-00000---000-0-0000-0-------00-\\
&-00-00---0-0-00000-0-0000-0----00-\\
&-----00--0--0-000000--0--0000-00-00-\\
&--00--0--0--000-0-----0-------0-\\
&---00-0000---0---0---0-00--0---000-\\
&00000-00------0--0--0-0---0--0---\\
&---000-00---00-0-0-0---0-00-0000-\\
&00---0-0-0--00-00-00-0--0-00-0000-\\
&-----0---00--0000------0-0-0-0---\\
&-00-0--000000--000--0-0--0-0-0-00---\\
&-0--00-0-0--0---000000---0000--0-\\
&00----0-0-0--000-0000-0--0-------0-\\
&-000-0--00-00-0-00-0---0-0-00-00-\\
&000------00------00---0-0-00000000-\\
&---0000-0-------0--0-0-0-0----0-\\
&0-0-00-0-0--0--00-00-0000-0--0--0-\\
&00-0--000000-0-00-00-0000---0000-000.
\end{aligned}
$$

More examples will be given in supplement materials or upon request.