In Search of Odd Perfect Numbers: A Computational Sandbox

M. R. DeDeo [0000-0003-0956-1421] and Matthew Thomas [0000-0001-6511-3223]

University of North Florida, Jacksonville, FL 32224, USA
mdedeo@unf.edu

Abstract. Several algorithmic approaches to finding odd perfect numbers have been attempted. This paper presents a novel open-access computational sandbox to search for odd perfect numbers for the purpose of making observations that could lead to new ways to search for OPNs and their properties. The development environment used, based on Object Pascal, allows easy modification of the software with cross-platform support for Windows, Mac, Linux, and mobile platforms. In addition, the paper discusses the most current restrictions on OPNs, other computational schemes and considers the difficulties described in working with large numbers.

Keywords: Odd Perfect Numbers, Computational Sandbox, Large Number Factorization

MSC: 11A25, 11A51

1 Introduction

Define $\sigma(N)$ to be the sum of the divisors of N. A number is considered *perfect* if $\sigma(N) = 2N$. The first perfect number is 6 as $\sigma(6) = 1 + 2 + 3 + 6 = 12 = 2(6)$. The abundancy number of a perfect number, which is defined to be the ratio $\sigma(N)/N$, is 2. Numbers for which this ratio is greater than (less than) 2 are called *abundant* (*deficient*) numbers.

There is a slew of conditions that, if an odd perfect number existed, would have to be satisfied. In 1888, Sylvester proved in a short, but clever, proof that an odd perfect number (OPN) must have three factors. He then subsequently proved an OPN must have at least four, and then five factors [16]. Define $\omega(n)$ to be the number of distinct prime divisors of n and $\Omega(n)$ to be the total number of factors. Other conditions include:

- *Form:*
  - $N = \wp^{\alpha} q_1^{2\beta_1} q_2^{2\beta_2} \dots q_k^{2\beta_k}$ where $\wp, q_1, q_2, \dots, q_k$ are distinct primes and $\wp \equiv \alpha \equiv 1 \bmod 4$ [6]
  - N is of the form $12K + 1$ or $36K + 9$ for some $K \in \mathbb{N}$ [18]

  - N is not divisible by 105 since 3, 5, and 7 cannot all divide N [13]
  - If for every β_i, $\beta_i \equiv 1 \bmod 3$, then N is not an OPN [10]
  - If for every β_i, $\beta_i \equiv 2 \bmod 5$ where $3|N$, then N is not an OPN [10]

- *Lower Bounds:*
  - For the largest prime factor component p, $p^{\alpha} > 10^{62}$ [15]
  - $\omega(N) \geq 9$ and $\omega(N) \geq 12$ if $3|N$ [14]
  - $\Omega(N) \geq 101$ [15]

- *Large Factors:*
  - $N > 10^{1500}$ [15]
  - Second and third largest primes are greater than 10,000 and 100, respectfully [9, 10]
- *Upper Bound:*
  - There are finitely many N with a fixed number of distinct prime factors [5]
  - The largest prime factor p of N is less than $(3N)^{1/3}$ [12]
  - The second largest prime factor p of N is less than $(2N)^{1/5}$ [9]
  - The smallest prime factor of N with all even powers lower than six is less than $exp(4.97401 \text{x} 10^{10})$ [20]
  - $N \leq 2^{4^{(k+1)}-2^{(k+1)}}$ [2]

Even before many of these conditions were determined, Sylvester insightfully wrote:

"...a prolonged mediation on the subject has satisfied me that the existence of any one such – its escape, so to say, from the complex web of conditions which hem it on all sides – would be little short of a miracle." [16]

2 Computational methods for OPN conditions

Several algorithmic approaches to finding OPNs have been attempted. We describe two as they offer insight into this project. In 2009, Nielsen implemented an algorithm in Mathematica that was different than previous approaches to confirm that N has at least nine distinct prime factors such that $\omega(N) \geq 9$ [14]. First, Nielsen fixed the bound B and did not let the bound increase with the algorithm. Instead, he increased it manually only if needed as allowing a computer to vary B automates the algorithm. Unfortunately, it does so at the expense of unnecessary complexity. Second, he used a lemma by Cohen [3] which allowed for stronger upper bounds on intervals for primes. Lastly, he compiled a list of 11 contradictions, four of which were different than those previously used.

Ochem and Rao used several algorithmic methods to navigate “roadblocks” that they uncover while trying to prove a lower bound for N [15] . They follow the approach of Brent et al. [1] with a method to by-pass roadblocks. With a minor modification of these techniques and factorization methods, they show that N must have at least 101 factors and that N must have a prime component p^{α} greater than 10^{62}.

To do this, they focus on the form of N itself, the total number of prime factors of N and its largest component. They then use factor chains which are constructed using branching to forbid certain factors. The algorithm then forbids every prime less than 10^{8} (as the largest prime factor must be greater than 10^{8}) using the following two contradictions: the abundancy number being strictly greater than 2 and the current number N has a prime power component greater than 10^{62}. Although this method can be extended past 10^{1500}, trying to search smarter proves to be difficult even with the ability to check for all prime factors less than one billion.

3 Considerations for computational OPN searches

Here we present a novel open-access computational sandbox to search for odd perfect numbers for the purpose of making observations that could lead to new ways to search for OPNs and their properties. The development environment used for the experiments described in this paper is Delphi which is based on Object Pascal which allows for easy modification of the software with cross-platform support for Windows, Mac, Linux, and mobile platforms.

Many computer languages have built-in integer data types that are limited to a maximum precision that is far too small to work with numbers as large as 10^{1500}. Without built-in support for large integers, as is the case with Delphi as of version 10.3, a third-party software library is needed. There are several options for big number libraries for Delphi, most of which are written in Object Pascal with inline assembly optimizations. Unfortunately, they are not optimized for multiple platforms. It is always desirable for software tools to be available on multiple platforms, but several Object Pascal libraries for big numbers which are optimized for one platform (e.g. Windows) perform very slowly on others (e.g. Mac OS and Linux). Another option is to use a shared library which has bindings available for Delphi and which has optimal performance for many platforms.

The *GNU Multiple Precision Arithmetic Library* (GMP) was ultimately chosen for this project due to its flexibility and ease-of-use across platforms. GMP is a free library which provides the implementation for arbitrary precision integer, rational, and floating-point arithmetic [7]. There is no practical limit to the precision, except for the limitations implied by the available memory in the machine GMP runs on. In addition, GMP has a rich set of functions and its functions have a regular interface.

In order to examine the values of $\sigma(N)/N$ in decimal form, a computer language also needs the ability to divide two extremely large integers and store the result as a floating-point number that may or may not need an extremely large precision. If our only interest is the integer part, e.g., we only want to know whether or not $\sigma(N)/N$ is greater than, equal to, or less than two, then there is no need to do any floating-point computations at all. If our only interest is a few (10 to 20) significant digits in the floating-point output, then built-in floating-point data types may suffice. However, if we want to compare two values of $\sigma(N)/N$ for different values of N, we need floating-point precision of hundreds or even thousands of significant digits to detect the difference. An example of this is described later where $\sigma(N)/N$ for different values of N have the same decimal digits for at least the first 200 digits.

4 Considerations for large number calculations

4.1 Computational Complexity

There are a large number of candidates for odd perfect numbers between 10^{1500} and 10^{1501} based on known properties of odd perfect numbers. Even when we find a candidate N, $\sigma(N)$ needs to be computed to verify that N, a very large number even for a computer, is perfect or not.

Using a large integer library, a program was written to implement a method which involves dividing N by every odd number from 3 to $\sqrt{N}$ and checking the remainder to see if those numbers are divisors. It was immediately realized that this would take a long time, thus the code was modified to do only one billion divisions and then measure the amount of time it took. This revealed that a typical processor such as an Intel Core i7, 10th gen, running at 1.3GHz can do approximately two million large integer divisions per second at about 30% CPU capacity using one core.

Even if the code were modified to be multi-threaded in order to use 100% of its CPU and further optimized, and assuming ten million divisions per second could be attained, it would still take about 1.5×10^{726} years to compute $\sigma(N)$ in this manner. This is based on $\sqrt{10^{1500}} \div 2 = 5 \times 10^{749}$divisions needed at a rate of 10^6 divisions per second with one billion computers being used to split the work.

4.2 Computing $\sigma(N)$

Using known formulas, computing $\sigma(N)$ takes less than a microsecond if we know the prime factorization of N. Performing a prime factorization of a larger number such as $N = 10^{1500} + 1$ based on the first 50,000,000 prime numbers can take about 30 seconds to execute with the following divisors, none of which has a power greater than one: 73, 137, 401, 1201, 1601, 24001, 32401, 1378001, 1676321, 31272001, 99990001, and 113850001. After dividing N by these numbers, the remaining factor is down to a number with 1443 digits.

Computing $\sigma(N)$ for the above known prime factors is easy but computing $\sigma(N)$ for the remaining factor may still be impossible with typical computing power. Using a list of all prime numbers with nine digits or less, the resulting prime factorization of $N = 10^{1500} + 1$ only found a few more prime factors. For additional numbers N, factoring N and determining its abundancy becomes even more complicated. Hence the creation of the sandbox.

5 Considerations for computational OPN searches

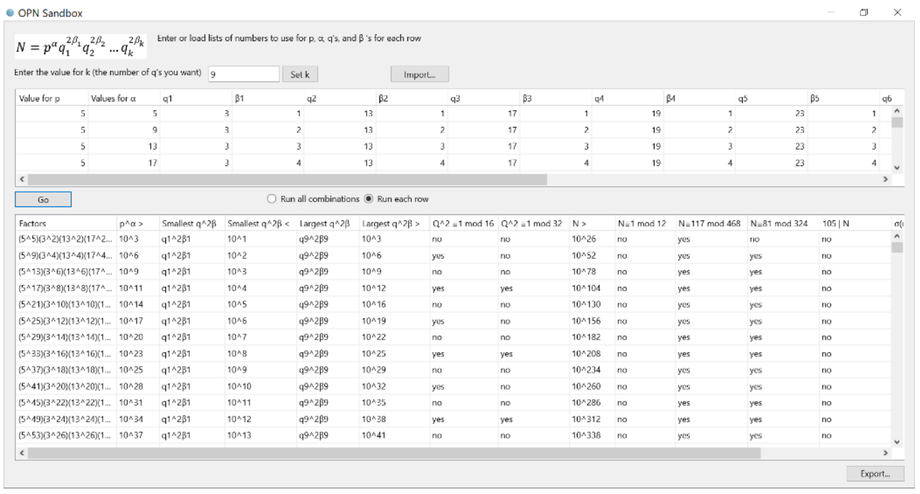
To address the difficulties described in working with large numbers, we created the OPN Sandbox (OPNS), a software tool which takes numbers already factored into prime powers and displays a grid of properties that can be exported and analyzed. The software's compiled executables for Windows and Mac OS are available on Github at [17]. There is also a command-line version for Windows, Mac OS, and Linux available.

OPN Sandbox provides a visual interface that lets us "play" with numbers and make observations. The heart of the software is the OPNS property processor (opnsprop) which, given the prime factorization of a number, calculates several properties of the number as they relate to OPNs. In addition to Linux, this functionality is built into the main application for Windows and Mac OS platforms. It is also available as a command-line tool named **opnsprop** on those platforms.

There are several ways in which OPNS can be used. The first way is to load a list of numbers, specified as prime powers, and have the OPN-related properties calculated for each and displayed in a grid. The list of numbers can be entered in an input grid or loaded via a comma-separated value (csv) file. As OPNS was being developed, the original intension was to use OPNS was to allow others to prepare a list of numbers and make them available to process. The expectation is that others can explore with their own ideas about methods of choosing the prime powers, and then their lists of numbers would be processed by OPNS to see what properties their numbers have. These lists can be prepared by any tool that is able to save a plain text csv file (for example, Excel, MATLAB, Python, etc.).

This is accomplished through the following process: each row of the input file represents the prime factors (and their powers) for a number with the form $N = \wp^{\alpha} q_1^{2\beta_1} q_2^{2\beta_2} \dots q_k^{2\beta_k}$. For example, let $N = 5^5 3^2 13^2 17^2 19^2 23^2 29^2 31^2 37^2 41^2$. An example of an input row is: 5, 5, 3, 1, 13, 1, 17, 1, 19, 1, 23, 1, 29, 1, 31, 1, 37, 1, 41, 1, where the first number is $\wp$, the second is α, and following pairs of numbers represent q and β for each prime component $q^{2\beta}$. This example has ten distinct prime factors, but OPNS can accept more with no fixed limit. The number of rows is only limited by available memory since OPNS loads the entire file into memory (although the command-line program opnsprop reads and process one row at a time, and thus has no limit on the file size it can process).

The software then processes each row and computes and displays various properties of each number. The properties checked are some of known properties of odd perfect numbers. Not all known properties of OPNs are provided but more will be added as the development of the software continues.


OPN Sandbox

$N = p^{\alpha} q_1^{2\beta_1} q_2^{2\beta_2} \ldots q_k^{2\beta_k}$ Enter or load lists of numbers to use for p, α, q's, and β 's for each row

Enter the value for k (the number of q's you want) 9 Set k Import...

Value for p	Values for α	q1	β1	q2	β2	q3	β3	q4	β4	q5	β5	q6
5	5	3	1	13	1	17	1	19	1	23	1	
5	9	3	2	13	2	17	2	19	2	23	2	
5	13	3	3	13	3	17	3	19	3	23	3	
5	17	3	4	13	4	17	4	19	4	23	4	

Go ○ Run all combinations ● Run each row

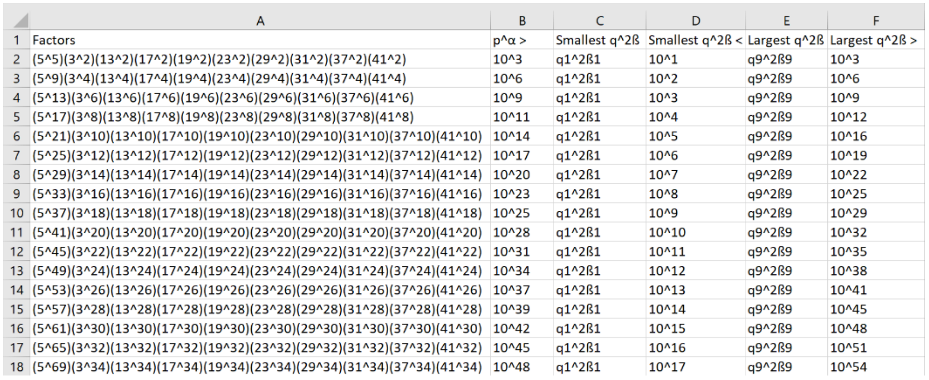
Factors	p^α >	Smallest q^2β	Smallest q^2β <	Largest q^2β	Largest q^2β >	Q^2 ≡1 mod 16	Q^2 ≡1 mod 32	N >	N≡1 mod 12	N≡117 mod 468	N≡81 mod 324	105 | N	σ(
(5^5)(3^2)(13^2)(17^2...	10^3	q1^2β1	10^1	q9^2β9	10^3	no	no	10^26	no	yes	no	no	
(5^9)(3^4)(13^4)(17^4...	10^6	q1^2β1	10^2	q9^2β9	10^6	yes	no	10^52	no	yes	yes	no	
(5^13)(3^6)(13^6)(17^...	10^9	q1^2β1	10^3	q9^2β9	10^9	no	no	10^78	no	yes	yes	no	
(5^17)(3^8)(13^8)(17^...	10^11	q1^2β1	10^4	q9^2β9	10^12	yes	yes	10^104	no	yes	yes	no	
(5^21)(3^10)(13^10)(1...	10^14	q1^2β1	10^5	q9^2β9	10^16	no	no	10^130	no	yes	yes	no	
(5^25)(3^12)(13^12)(1...	10^17	q1^2β1	10^6	q9^2β9	10^19	yes	no	10^156	no	yes	yes	no	
(5^29)(3^14)(13^14)(1...	10^20	q1^2β1	10^7	q9^2β9	10^22	no	no	10^182	no	yes	yes	no	
(5^33)(3^16)(13^16)(1...	10^23	q1^2β1	10^8	q9^2β9	10^25	yes	yes	10^208	no	yes	yes	no	
(5^37)(3^18)(13^18)(1...	10^25	q1^2β1	10^9	q9^2β9	10^29	no	no	10^234	no	yes	yes	no	
(5^41)(3^20)(13^20)(1...	10^28	q1^2β1	10^10	q9^2β9	10^32	yes	no	10^260	no	yes	yes	no	
(5^45)(3^22)(13^22)(1...	10^31	q1^2β1	10^11	q9^2β9	10^35	no	no	10^286	no	yes	yes	no	
(5^49)(3^24)(13^24)(1...	10^34	q1^2β1	10^12	q9^2β9	10^38	yes	yes	10^312	no	yes	yes	no	
(5^53)(3^26)(13^26)(1...	10^37	q1^2β1	10^13	q9^2β9	10^41	no	no	10^338	no	yes	yes	no	

Export...

Fig. 1. Visual interface of the OPN Sandbox.

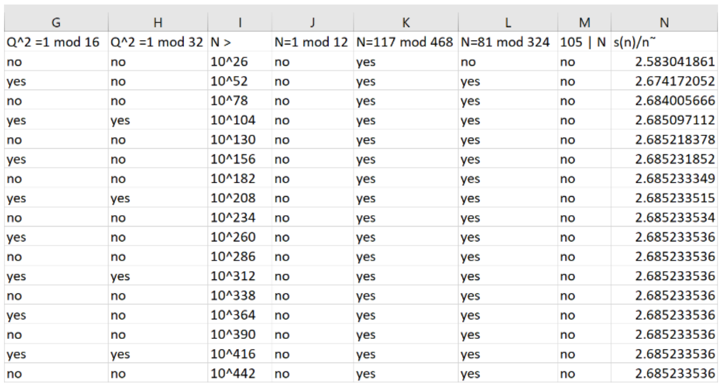
Figures 2a and 2b show an example of the output as displayed in Excel. The properties are shown in the header of each column. The output can then be exported as a plain text csv file that can be analyzed by another software tool. One of the limitations of Excel is that it is difficult to format the numbers to show more than a few (10 to 20) decimal places.

The output file exported by OPNS, viewed in a text file viewer, shows that the last column includes up to 200 significant digits for $\sigma(N)/N$, but these values are rounded by Excel.



	A	B	C	D	E	F
1	Factors	p^α >	Smallest q^2ß	Smallest q^2ß <	Largest q^2ß	Largest q^2ß >
2	(5^5)(3^2)(13^2)(17^2)(19^2)(23^2)(29^2)(31^2)(37^2)(41^2)	10^3	q1^2ß1	10^1	q9^2ß9	10^3
3	(5^9)(3^4)(13^4)(17^4)(19^4)(23^4)(29^4)(31^4)(37^4)(41^4)	10^6	q1^2ß1	10^2	q9^2ß9	10^6
4	(5^13)(3^6)(13^6)(17^6)(19^6)(23^6)(29^6)(31^6)(37^6)(41^6)	10^9	q1^2ß1	10^3	q9^2ß9	10^9
5	(5^17)(3^8)(13^8)(17^8)(19^8)(23^8)(29^8)(31^8)(37^8)(41^8)	10^11	q1^2ß1	10^4	q9^2ß9	10^12
6	(5^21)(3^10)(13^10)(17^10)(19^10)(23^10)(29^10)(31^10)(37^10)(41^10)	10^14	q1^2ß1	10^5	q9^2ß9	10^16
7	(5^25)(3^12)(13^12)(17^12)(19^12)(23^12)(29^12)(31^12)(37^12)(41^12)	10^17	q1^2ß1	10^6	q9^2ß9	10^19
8	(5^29)(3^14)(13^14)(17^14)(19^14)(23^14)(29^14)(31^14)(37^14)(41^14)	10^20	q1^2ß1	10^7	q9^2ß9	10^22
9	(5^33)(3^16)(13^16)(17^16)(19^16)(23^16)(29^16)(31^16)(37^16)(41^16)	10^23	q1^2ß1	10^8	q9^2ß9	10^25
10	(5^37)(3^18)(13^18)(17^18)(19^18)(23^18)(29^18)(31^18)(37^18)(41^18)	10^25	q1^2ß1	10^9	q9^2ß9	10^29
11	(5^41)(3^20)(13^20)(17^20)(19^20)(23^20)(29^20)(31^20)(37^20)(41^20)	10^28	q1^2ß1	10^10	q9^2ß9	10^32
12	(5^45)(3^22)(13^22)(17^22)(19^22)(23^22)(29^22)(31^22)(37^22)(41^22)	10^31	q1^2ß1	10^11	q9^2ß9	10^35
13	(5^49)(3^24)(13^24)(17^24)(19^24)(23^24)(29^24)(31^24)(37^24)(41^24)	10^34	q1^2ß1	10^12	q9^2ß9	10^38
14	(5^53)(3^26)(13^26)(17^26)(19^26)(23^26)(29^26)(31^26)(37^26)(41^26)	10^37	q1^2ß1	10^13	q9^2ß9	10^41
15	(5^57)(3^28)(13^28)(17^28)(19^28)(23^28)(29^28)(31^28)(37^28)(41^28)	10^39	q1^2ß1	10^14	q9^2ß9	10^45
16	(5^61)(3^30)(13^30)(17^30)(19^30)(23^30)(29^30)(31^30)(37^30)(41^30)	10^42	q1^2ß1	10^15	q9^2ß9	10^48
17	(5^65)(3^32)(13^32)(17^32)(19^32)(23^32)(29^32)(31^32)(37^32)(41^32)	10^45	q1^2ß1	10^16	q9^2ß9	10^51
18	(5^69)(3^34)(13^34)(17^34)(19^34)(23^34)(29^34)(31^34)(37^34)(41^34)	10^48	q1^2ß1	10^17	q9^2ß9	10^54

Fig. 2a. Example of the OPN Sandbox factorization output (Columns A through F) displaying the factors of each N and its properties



G	H	I	J	K	L	M	N
Q^2 =1 mod 16	Q^2 =1 mod 32	N >	N=1 mod 12	N=117 mod 468	N=81 mod 324	105 | N	s(n)/n˜
no	no	10^26	no	yes	no	no	2.583041861
yes	no	10^52	no	yes	yes	no	2.674172052
no	no	10^78	no	yes	yes	no	2.684005666
yes	yes	10^104	no	yes	yes	no	2.685097112
no	no	10^130	no	yes	yes	no	2.685218378
yes	no	10^156	no	yes	yes	no	2.685231852
no	no	10^182	no	yes	yes	no	2.685233349
yes	yes	10^208	no	yes	yes	no	2.685233515
no	no	10^234	no	yes	yes	no	2.685233534
yes	no	10^260	no	yes	yes	no	2.685233536
no	no	10^286	no	yes	yes	no	2.685233536
yes	yes	10^312	no	yes	yes	no	2.685233536
no	no	10^338	no	yes	yes	no	2.685233536
yes	no	10^364	no	yes	yes	no	2.685233536
no	no	10^390	no	yes	yes	no	2.685233536
yes	yes	10^416	no	yes	yes	no	2.685233536
no	no	10^442	no	yes	yes	no	2.685233536

Fig. 2b. Example of the OPN Sandbox factorization output (Columns G through N) displaying each *N*'s satisfaction of known conditionals

The second way OPNS can be used builds upon the first process: First, one or more numbers can be entered or loaded in the input grid (the upper grid shown in Figure 1) in prime power form, and the OPN-related properties are shown (the lower grid in Figure 1). We can then adjust the numbers by changing the prime factors (or exponents) and see instant feedback in the re-calculation of the properties. The keyboard arrow keys can be used to quickly move from one input grid cell to another. The number in the selected cell can be edited directly. Additionally, keyboard shortcuts can be used to increase or decrease the number in the grid cell in a way appropriate for the type of number. If the number is in the column for $\wp$, the number is changed (increased or decreased, depending on the keyboard shortcut pressed) so that the new number is a prime number not already used as any of the other prime factors in that grid row and is of the form $4K + 1$. For α, the new number is also of the form $4K + 1$. For any of the q's, the new number is a prime number not already used as a prime factor in that grid row. When OPNS selects the prime numbers in response to keyboard shortcuts, it generates the numbers using a simple prime number generator. A video description of the paper and the process can be found online at [5].

There is another option that is useful when the prime numbers become large enough that to generate the next prime number becomes noticeably sluggish. OPNS can load a large pre-generated list of primes from a database. The database contains all prime numbers which are 12 digits or less, however OPNS only loads a small portion (1000 numbers) at a time as needed. This option allows using keyboard shortcuts repeatedly with almost no delay.

As we use OPNS to play with numbers and make observations, this leads to ideas for further experiments. If an experiment involves an algorithm which is to be automated, then another way to use OPNS is as the visual interface for the algorithm after it has been coded and made part of OPNS. This allows the algorithm to utilize any of the features already built into OPNS. The algorithm has access to the routines which calculate the OPN-related properties, access to the prime number database, generates log files, and provides access to the routines which determines how to increase or decrease prime factors.

5.1 Observations from Sample Data

The sample input data used to generate the output in Figures 1, 2a and 2b are available in the Github repository for OPNS [17]. Although the data imported to generate Figure 1 was arbitrarily selected, some interesting observations can be made from the output.

2.583041860510535108036352002877071435795249326261

2.**6**74172051623086656780281505347020612394456779269

2.**68**4005666081849539671557283926575510653299892257

2.**685**097111613254595822198478816780005837171143754

2.**6852**18377673571768746477774497067200811200508912

2.**68523**1851650193694322547042171733263972934384087

2.**685233**348758539196633488837962439494180024705543

2.**6852335**15103909955198676402476144779685264815363

2.**68523353**3586728922765512601519184075441188654620

2.**685233535**640375474684518751937993291961253132068

2.**6852335358**68558424897548225783421693389501126878

2.**68523353589**3912086032328137307847803117829144050

2.**68523353589**6729159491748120736567156007128517841

2.**6852335358970**42167653905896633245011386715702384

2.**6852335358970**76946338590093954862460801700837669

2.**68523353589708**0810636888338101707450169001503772

2.**6852335358970812**40003365920784690218518977333048

2.**6852335358970812**87710752318860577192731295836611

2.**68523353589708129**3011573029757897967643486990738

Fig. 3. OPN Sandbox abundancy number from data in Fig. 1.

Using the input data generated by an Excel spreadsheet starting with the row mentioned in the previous section: 5, 5, 3, 1, 13, 1, 17, 1, 19, 1, 23, 1, 29, 1, 31, 1, 37, 1, 41, 1, another 499 rows were generated by incrementing α by 4 and incrementing all other powers by 1 from one row to the next.

One pattern observed is that the two properties $Q^2 \equiv 1 \bmod 16$ and $Q^2 \equiv 1 \bmod 32$ where $Q^2 = q_1^{2\beta_1} \ldots q_k^{2\beta_k}$ is the perfect square portion of an odd perfect number, has the repeated pattern: no/no, yes/no, no/no, yes/yes. Although we leave it for the reader to explore the fairly easy explanation for this pattern given the form of the prime factors of *N*, this observation demonstrates how patterns can be revealed.

Another interesting observation related to the data in Figure 1, which cannot be seen in Excel, but is displayed in Figure 3. The last column of the Excel output displays the abundancy number, $\sigma(N)/N$. Notice that the list in Figure 3 has an increasing number of decimal digits that are identical. Beyond the 207th row, at least 200 decimal digitals repeat from one output row to the next. Again, another interesting observation which can be explained by values of the ratio of $\sigma(q_i^{2\beta_i})/q_i^{2\beta_i}$ as β_i increases for a fixed q.

5.2 Observations from playing with the numbers

While testing OPNS with one row of basic input, it is very easy to quickly reach a point where we observer the transition of $\sigma(N)/N$ from above 2 to below 2, and vice versa. We again start with the basic input corresponding to the number above:

$$N = \wp^{\alpha} q_1^{2\beta_1} q_2^{2\beta_2} \dots q_{11}^{2\beta_{11}} = 5^1 3^4 11^2 13^2 17^2 19^2 23^2 29^2 31^2 37^2 41^2 43^2 \,.$$

Note that this N is just a basic product of prime powers consisting of the distinct prime factors that are consecutive from 3 through 41 (skipping 7), with small exponents for the non-special primes (2 and 4). Changing the lower prime factors has a big effect on $\sigma(N)/N$, but not as much change when the exponents are changed (as was previously observed). We keep the two lowest prime factors, $\wp$ and $q_1^{\beta_1}$, and keep all of the exponents fixed. Then, starting with q_2 , we replace 11 with 47, the next prime factor not already used. We then we replace 13 with 53, the next prime factor not already used, and so on. Each time, we observe that $\sigma(N)/N$ decreases from approximately 2.89 to 2.68, then to 2.53, and so on. By the time we get to q_{11} and replace 43 with 89, $\sigma(N)/N$ has reached approximately 2.09.

Making a second pass starting again with q_2 , replace 47 with 97 and repeat the process. By the time we reach q_7 and replace it, $\sigma(N)/N$ is now approximately 2.007. One more replacement, q_8 , causes $\sigma(N)/N$ to go below 2. Replacing the prime factors in this way amounts to a shifting frame of consecutive prime numbers as demonstrated in Table 3 (with $q_2^{2\beta_2} \dots q_{11}^{2\beta_{11}}$ sorted).

These and other observations led to an idea for an algorithm with the goal of finding numbers whose abundancy numbers get closer and closer to 2 and then making observations from that experiment. Another algorithm is under development and new ideas are surfacing which may lead to further revisions, but so far, the current algorithm has found numbers on the deficient side with $\sigma(N)/N \approx 1.999999999479$. The number itself, which is greater than 10^{5442}, has 704 distinct prime factors. On the abundant side, another number N was found where $\sigma(N)/N \approx 2.000000000134$.

All of the approximations mentioned are actually displayed and logged by OPNS to a precision of 200 decimal digits (which is the default precision of the OPN property processor). This is a setting that it passes along to the GMP library which, as previously mentioned, can be configured with a precision of any degree only limited by available memory.

Table 3. Shifting frame of prime powers.

$\wp^{\alpha}$	$q_1^{2\beta_1}$						$q_2^{2\beta_2} \dots q_{11}^{2\beta_{11}}$								$\sigma(N)/N$
5^1	3^4	11^2	13^2	17^2	19^2	23^2	29^2	31^2	37^2	41^2	43^2				2.89
5^1	3^4		13^2	17^2	19^2	23^2	29^2	31^2	37^2	41^2	43^2	47^2			2.68
5^1	3^4			17^2	19^2	23^2	29^2	31^2	37^2	41^2	43^2	47^2	53^2		2.53
5^1	3^4				19^2	23^2	29^2	31^2	37^2	41^2	43^2	47^2	53^2	59^2	2.42

6 Future software development

We mentioned several observations which led to ideas for a revised algorithm that is still in development. In the process of running the new algorithm, observations have already led to new ideas additional modifications. We note that, in the scheme of things, the existing algorithm looks at a relatively small set of numbers.
For example, to find the number N previously mentioned where $\sigma(N)/N \approx 1.999999999479$, the algorithm examined only millions of numbers among more than 10^{5442} numbers. Although the existing algorithm is purely automatic, we can expect to move toward a machine learning algorithm to search for OPNs since intelligent decisions made by the algorithm can be used to decide which numbers to examine.

In order to encourage others to explore this interesting problem, we note that OPN Sandbox is written with the freely available Delphi Community Edition which is based on Object Pascal. This makes it easy, even by novice coders, to modify and customize the code. Delphi is capable of compiling a single base of source code into cross-platform visual applications with support for Windows, Mac OS, iOS, Android, and Linux. The portion of the OPNS code that makes up the OPN property processor is written to not only compile under Delphi but also to compile by the Free Pascal Compiler (FPC) which supports even more platforms. Both Delphi and FPC have available libraries to make it easy to write applications which communicate over a network. This choice conveniently sets programmers up to enable our applications and algorithms to run as distributed processes among a variety of devices.

We also note that the existing algorithm lends itself to run as a distributed process which opens the possibility to make it faster. As a next step, the greatest importance is the ability to run a machine learning algorithm that can be distributed to accomplish in a reasonable amount of time what a single-threaded process would accomplish in many weeks or months.

References

1. Brent, R.P., Cohen, G.L., te Reile, H.J.J.: Improved techniques for lower bounds for odd perfect numbers. Math. Comp. 57(196)**,** 857-868 (1991).
2. Chen, Y.-G., Tang, C.-E: Improved upper bounds for odd multiperfect numbers. Bull. of the Australian Math. Soc. 89(3), 353-359 (2014).
3. Cohen, G.L.: On odd perfect numbers, Fibonacci Q. 16, 523–527 (1978).
4. DeDeo, Michelle, and Matthew Thomas. “In Search of Odd Perfect Numbers: A Computational Sandbox.” YouTube, 12 May 2021, youtu.be/YFytb_5q7m0.
5. Dickson, L. E. History of the Theory of Numbers, Vol. 1: Divisibility and Primality. Reprint, Dover, New York (2005).
6. Dunham, W., Euler, L: Euler: The Master of Us All. Math. Assn. of America, Washington, DC (1999).
7. The GNU MP Bignum Library, 2020, gmplib.org/.
8. Goto, T., Ohno, Y.: Odd perfect numbers have a prime factor exceeding 10^8. Mathematics of Computation. **77**(263), 1859–1868 (2008).
9. Iannucci, D. E.: The Second Largest Prime Divisor of an Odd Perfect Number Exceeds Ten Thousand. Math. Comput. 68(228), 1749-1760 (1999).
10. Iannucci, D. E.: The Third Largest Prime Divisor of an Odd Perfect Number Exceeds One Hundred. Math. Comput. 69(230), 867-879 (2000).
11. Iannucci, D. E. and Sorli, M.: On the Total Number of Prime Factors of an Odd Perfect Number. Math. Comp. 72(244), 2077-2085 (2003).
12. Konyagin, S., Acquaah, P.: On Prime Factors of Odd Perfect Numbers. Intl. Journal of Number Theory 8(6), 1537–1540 (2012).
13. Kühnel, U.: Verschärfung der notwendigen Bedingungen für die Existenz von ungeraden vollkommenen Zahlen. Mathematische Zeitschrift. 52, 201–211(1949).
14. Nielsen, P.: Odd Perfect Numbers Have at Least Nine Distinct Prime Factors. Math. of Comput., 76(260), 2109–2127 (2007).
15. Ochem, P. and Rao, M.: Odd Perfect Numbers Are Greater than 10^{1500}, Math. Comput. 81(279), 1869-1877 (2012).
16. Sylvester, J.J.: Mathematical Papers*,* Vol. 4, Reprint, Chelsea, New York (1973).
17. Thomas, Matthew. “Mathprojects/OPNSandbox.” GitHub, github.com/mathprojects/OPNSandbox.
18. Touchard, J.: On Prime Numbers and Perfect Numbers. Scripta Math. 19, 35-39 (1953).
19. Velthuis, Rudy: Mathprojects/DelphiBigNumbers. *GitHub*, github.com/mathprojects/DelphiBigNumbers.
20. Yamada, T. "On the Divisibility of Odd Perfect Numbers by a High Power of a Prime." 16 Nov 2005. https://arxiv.org/abs/math.NT/0511410.