A technical approach to finding primes within a limited boundary

Junho Eom[*]

Department of Zoology, University of British Columbia, Vancouver, BC, V6T 1Z4, Canada

* Corresponding Author

Junho Eom

Dept. of Zoology, University of British Columbia, Vancouver, B.C., Canada V6T 1Z4

Email: zuno3302@gmail.com

ORCID ID:

Junho Eom 0000-0003-0278-0082

Abstract

At least one prime less than n ($n \geq 2$) is known to be used as a factor for composites between n and n^2, and this is explained by prime wave analysis. In this paper, the prime wave analysis is modified with a modular operator and applied to finding new primes within a limited boundary. In results, using the known primes less than 3, the composites were eliminated, and collected remaining prime candidates within a limited boundary between 3 and 3^2. The boundary was sequentially extended from 3^2 to 9^2, 81^2, and 6561^2 by finding 2, 18, 825, and 2606318 prime candidates; these candidates were verified as new primes using the using online databases. In addition, the boundary was extended from 6561^2 to 43046721^2 and the serial new primes were also found within a randomly selected boundary between 6561^2 and 43046721^2. In general, it was concluded that the prime wave analysis modified with a modular operator could be a practical technique for finding new primes within a limited boundary.

1. Introduction

The Sieve of Eratosthenes is a well-known algorithm for finding primes up to any given integer by removing composites with the prime numbers (Stein 2000). For instance, using the first prime 2, it is possible to obtain the next prime 3 by eliminating multiples of 2. Similarly, multiples of 3 can also be eliminated when obtaining the subsequent primes up to any given number.

The Sieve of Eratosthenes is modified by the prime wave analysis, which defines that the series of prime waves less than integer n are directly connected to composites. The passively remaining numbers, which are not affected by the continued known prime waves, are all primes within a limited boundary between n and n^2 (Eom 2024a). As new primes result from the relationship between the known primes and the composites, the known primes less than n and the new primes within a limited boundary between n and n^2 have a cause-and-effect relationship. If the boundary is limited to $2n$ instead of n^2, the cause-and-effect relationship remains, and the primes less than n and the new primes between n and $2n$ are thought a form the partial symmetry (Guiasu 2019) due to the asymmetrical relationship between the primes and the composites; this is thought to satisfy Goldbach's conjecture (Eom 2024b). Therefore, developing a practical technique for finding new primes using the prime wave analysis is helpful in understanding the structure of primes, which results from the cause-and-effect relationship among the known primes, the composites, and the new primes; the knowledge background of prime structure can be applied to explain the prime related conjectures and problems.

The purpose of this paper is to find primes within a limited boundary using the prime wave analysis. In the prime wave analysis, the value was defined by irrational numbers, making it difficult to handle. Thus, the prime wave was modified using a modular operator in C^{++} and tested with positive integers to find real primes by extending the boundary from 3^2 to 43046721^2 (Figure 1); the accuracy was verified using an online database.

2. Materials and methods

2.1. Minimum requirements for finding primes using a modular operator

The serial known primes less than n saved in *input.txt* was used to eliminate the composites within a limited boundary between n and n^2. The remaining numbers were exported to *ouput.txt*. Thus, the modified prime wave analysis using a modular operator required three minimum elements.

1) A document (*input.txt*) saved serial known primes less than n.
2) Software that enables the use of the modular operator to find the prime candidates within a limited number boundary between n and n^2.
3) A document (*output.txt*) to export the prime candidates.

Regarding the software for operating the modular operator, Visual Studio 2022 (Microsoft Corporation, Washington, US) was used to minimize the impact on processing speed with C^{++} code. The exported prime candidates in *output.txt* were matched with known prime numbers using online databases.

- Number Empire (numberempire.com) – used for verifying primes over 10^{15}
- The Prime Pages: prime number research and records (t5k.org) – used for verifying primes up to 10^{13}
- Prime I.T. (compoasso.free.fr) – used for verifying primes up to 10^{9}
- Free Online Calculators (calculator.net) – used for calculation over 10^{10}.

The computer specifications to run Visual Studio 2022 were listed.

- Processor – AMD Ryzen 7 2700X Eight-Core Processor, 3.70 GHz
- Installed RAM – 16.0 GB
- System operator – Windows 10 Pro
- System type – 64-bit operating system, x64-based processor.

2.2. Modular operator setup with C^{++} in Visual Studio 2022

Prior to setting up the modular operator, the variable *'isDivisible'* was defined with *'number'* and *'divisor'*. The variable *'number'* represented odd numbers within a specified boundary, while *'divisor'* represented known serial primes. Using these defined variables, the modular operator *'number % divisor == 0'* was prepared.

```
bool isDivisible(const long long number, const long long divisor)
{
   return (number % divisor == 0);
}
```

The known serial primes were imported from *input.txt* using the *'ifstream'* function and vectorized to increase the speed of the computation process. Using the *'ofstream'* function, *output.txt* was also prepared to export the prime candidates.

```
 std::ifstream inputFile("input.txt");
 std::ofstream oFile;
 oFile.open("output.txt");


  // Read the serial primes in input.txt into a vector
std::vector<long long> inputValues;
long long inputValue;
while (inputFile >> inputValue) {
   inputValues.push_back(inputValue);
}
inputFile.close();
```

The minimum and maximum boundaries were designed to be manually entered. Later, the prime candidates were searched between *'startRange'* and *'endRange'*.

```
long long startRange, endRange;

// Manually input the minimum and maximum boundaries
std::cout << "Enter the odd minimum number boundary: ";
std::cin >> startRange;

std::cout << "Enter the maximum number boundary: ";
std::cin >> endRange;
```

Within a manually entered boundary, the value *'startRange'* was sequentially increased by adding 2 until it reached the value of *'endRange'*. Therefore, the manually entered minimum number boundary should always be odd. The odd numbers were defined as *'currentNum'* and considered as prime candidates. If *'currentNum'* could be divided by *'divisor'* imported from *input.txt*, then *'currentNum'* was eliminated; otherwise, it was exported to *output.txt*.

```
// Generate prime candidates within a limited boundary between startRange and endRange

 for (long long currentNum = startRange; currentNum <= endRange; currentNum += 2) {
        bool isDivisibleByAny = false;
        // Check whether the prime candidates are divided by divisor imported from input.txt
        for (const long long divisor : inputValues) {
            if (isDivisible(currentNum, divisor)) {
                isDivisibleByAny = true;
                break;
            }
        }

        // If the prime candidates are not divided by divisor, then export them to output.txt
        if (!isDivisibleByAny) {
            std::cout << currentNum << std::endl;
            oFile << currentNum << std::endl;
        }
```

Overall, the completed code for finding new primes within a limited number boundary was listed.

```
#include <fstream>
#include <iostream>
#include <vector>

bool isDivisible(const long long number, const long long divisor)
{
    return (number % divisor == 0);
}

int main()
{
    std::ifstream inputFile("input.txt");
    std::ofstream oFile;
    oFile.open("output.txt");
    long long startRange, endRange;

    // Read the list of primes in input.txt into a vector
    std::vector<long long> inputValues;
    long long inputValue;
    while (inputFile >> inputValue) {
       inputValues.push_back(inputValue);
    }
    inputFile.close();

    // Manually input the minimum and maximum number boundaries
    std::cout << " Enter the odd minimum number boundary: ";
    std::cin >> startRange;
    std::cout << " Enter the maximum number boundary: ";
    std::cin >> endRange;

    // Generate prime candidates within a limited boundary between startRange and endRange
    for (long long currentNum = startRange; currentNum <= endRange; currentNum += 2) {
          bool isDivisibleByAny = false;

          // Check whether the prime candidates are divided by divisor imported from input.txt
          for (const long long divisor : inputValues) {
             if (isDivisible(currentNum, divisor)) {
                isDivisibleByAny = true;
                break;
             }
          }

      // If the prime candidates are not divided by divisor, then export them to output.txt
          if (!isDivisibleByAny) {
             std::cout << currentNum << std::endl;
             oFile << currentNum << std::endl;
          }
    }
    oFile.close();
    return 0;
}
```

3. Results

Using the known serial primes less than 3, the number boundary was initially limited to between 3 and 3^2 and systematically extended to 6561^2, finding 2606318 prime candidates. The number boundary was further extended from 6561^2 to 43046721^2, and partial serial prime candidates were also found within any selected boundary between 6561^2 and 43046721^2. All prime candidates were matched in the online databases, and 100% accuracy was verified.

3.1. Finding new primes within a limited number boundary between 3^2 and 6561^2

Example 1. The known primes in *input.txt* were 2 and 3. These primes were used as factors to eliminate composites within a limited boundary between 3 and 3^2. As a result, all composites were eliminated, leaving only the remaining numbers expected to be new primes.

• The primes given in *input.txt*: 2, 3

• The manually entered number boundary:

- Enter the odd minimum number boundary: 3

- Enter the maximum number boundary: 3^2 or 9

• Prime candidates exported to *output.txt*: 5, 7

• Total number of prime candidates in *output.txt*: 2

• The accuracy between prime candidates and primes in the online database: 100%

Example 2. The newly confirmed primes in *output.txt* were added to *input.txt*; as a result, the boundary was extended but limited between 9 and 9^2. All composites were eliminated, leaving only the remaining numbers expected to be primes.

• The primes given in *input.txt*: 2, 3, 5, 7

• The manually entered number boundary:

- Enter the odd minimum number boundary: 9

- Enter the maximum number boundary: 9^2 or 81

• Prime candidates exported to *output.txt*:

11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79

• Total number of prime candidates in *output.txt*: 18

• The accuracy between prime candidates and primes in the online database: 100%

Example 3. The newly confirmed primes in *output.txt* were added to *input.txt*; as a result, the boundary was extended but limited between 81 and 81^2. All composites were eliminated, leaving only the remaining numbers expected to be primes.

• The primes given in *input.txt*:

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79

• The manually entered number boundary:

- Enter the odd minimum number boundary: 81

- Enter the maximum number boundary: 81^2 or 6561

• Prime candidates exported to *output.txt*:

83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283,

…

6287, 6299, 6301, 6311, 6317, 6323, 6329, 6337, 6343, 6353, 6359, 6361, 6367, 6373, 6379, 6389, 6397, 6421, 6427, 6449, 6451, 6469, 6473, 6481, 6491, 6521, 6529, 6547, 6551, 6553

• Total number of prime candidates in *output.txt*: 825

• The accuracy between prime candidates and primes in the online database: 100%

Example 4. The newly confirmed primes in *output.txt* were added to *input.txt*; as a result, the boundary was extended but limited between 6561 and 6561^2. All composites were eliminated, leaving only the remaining numbers expected to be primes.

• The primes given in *input.txt*:

2, 3, 5, 7, 11, 13, 17, 19, 23, 31, 37, …, 6549, 6521, 6529, 6547, 6551, 6553

• The manually entered number boundary:

- Enter the odd minimum number boundary: 6561
- Enter the maximum number boundary: 6561^2 or 43046721

• Prime candidates exported to *output.txt*:

6553, 6563, 6569, 6571, 6577, 6581, 6599, 6607, 6619, 6637, 6653, 6659, 6661, 6673, 6679, 6689, 6691, 6701, 6703, 6709, 6719, 6733, 6737, 6761, 6763, 6779, 6781, 6791, 6793, 6803,

…

43046309, 43046323, 43046371, 43046401, 43046467, 43046477, 43046537, 43046543, 43046579, 43046581, 43046587, 43046599, 43046603, 43046611, 43046617, 43046623

• Total number of prime candidates in *output.txt*: 2606318
• The accuracy between prime candidates and primes in the online database: 100%

• Running time: 491.098 sec (~ 8.2 min)

3.2. Finding serial new primes within selected any boundary between 43046721 and 43046721^2

Example 5. The newly confirmed primes in *output.txt* were added to *input.txt*; as a result, all composites were eliminated, leaving only remaining numbers expected to be primes between 1234567891224567 and 1234567891234567.

• The primes given in *input.txt*:

2, 3, 5, 7, 11, 13, 17, 19, 23, 31, 37, … , 43046611, 43046617 ,43046623

• The manually entered any number boundary between 43046721 and 43046721^2:

- Enter the odd minimum number boundary: 1234567891224567
- Enter the maximum number boundary: 1234567891234567

• Prime candidates exported to *output.txt*:

1234567891224581, 1234567891224643 ,1234567891224839, 1234567891224851,
1234567891224883, 1234567891224917, 1234567891224919, 1234567891224949,
1234567891224961, 1234567891225037, 1234567891225063, 1234567891225069,
1234567891225099, 1234567891225133, 1234567891225187, 1234567891225247,
1234567891225283, 1234567891225343, 1234567891225349, 1234567891225367,
1234567891225397, 1234567891225433, 1234567891225463, 1234567891225511,
1234567891225519, 1234567891225567, 1234567891225573, 1234567891225609,
1234567891225667, 1234567891225679, 1234567891225751, 1234567891225909,
1234567891225927, 1234567891225951, 1234567891225957, 1234567891226053,
1234567891226093, 1234567891226099, 1234567891226119, 1234567891226129,
1234567891226167, 1234567891226249, 1234567891226281, 1234567891226291,
1234567891226309, 1234567891226323, 1234567891226333, 1234567891226339,
1234567891226353, 1234567891226393, 1234567891226497, 1234567891226507,
1234567891226561, 1234567891226653, 1234567891226707, 1234567891226773,
1234567891226777, 1234567891226881, 1234567891226963, 1234567891226977,
1234567891226993, 1234567891227023, 1234567891227113, 1234567891227161,
1234567891227173, 1234567891227271, 1234567891227331, 1234567891227389,
1234567891227403, 1234567891227427, 1234567891227439, 1234567891227449,
1234567891227487, 1234567891227521, 1234567891227571, 1234567891227607,
1234567891227623, 1234567891227631, 1234567891227757, 1234567891227791,
1234567891227809, 1234567891227821, 1234567891227857, 1234567891227859,
1234567891227863, 1234567891227877, 1234567891227899, 1234567891227907,
1234567891227919, 1234567891228039, 1234567891228079, 1234567891228127,
1234567891228141, 1234567891228157, 1234567891228223, 1234567891228249,
1234567891228271, 1234567891228319, 1234567891228327, 1234567891228403,
1234567891228423, 1234567891228433, 1234567891228471, 1234567891228513,
1234567891228519, 1234567891228541, 1234567891228543, 1234567891228597,
1234567891228601, 1234567891228657, 1234567891228663, 1234567891228703,
1234567891228751, 1234567891228753, 1234567891228787, 1234567891228799,
1234567891228801, 1234567891228877, 1234567891228933, 1234567891228937,
1234567891228979, 1234567891228981, 1234567891229003, 1234567891229009,
1234567891229083, 1234567891229111, 1234567891229137, 1234567891229143,
1234567891229177, 1234567891229249, 1234567891229269, 1234567891229339,
1234567891229363, 1234567891229371, 1234567891229429, 1234567891229447,

1234567891229593, 1234567891229627, 1234567891229657, 1234567891229713,
1234567891229719, 1234567891229723, 1234567891229731, 1234567891229789,
1234567891229791, 1234567891229797, 1234567891229807, 1234567891229833,
1234567891229837, 1234567891229839, 1234567891229917, 1234567891229921,
1234567891230019, 1234567891230041, 1234567891230053, 1234567891230077,
1234567891230131, 1234567891230187, 1234567891230203, 1234567891230313,
1234567891230407, 1234567891230413, 1234567891230419, 1234567891230469,
1234567891230511, 1234567891230539, 1234567891230551, 1234567891230563,
1234567891230601, 1234567891230649, 1234567891230689, 1234567891230721,
1234567891230749, 1234567891230767, 1234567891230787, 1234567891230803,
1234567891230809, 1234567891230899, 1234567891230907, 1234567891230911,
1234567891230941, 1234567891230991, 1234567891231021, 1234567891231057,
1234567891231061, 1234567891231091, 1234567891231139, 1234567891231163,
1234567891231177, 1234567891231223, 1234567891231231, 1234567891231247,
1234567891231303, 1234567891231319, 1234567891231331, 1234567891231387,
1234567891231531, 1234567891231547, 1234567891231553, 1234567891231601,
1234567891231637, 1234567891231687, 1234567891231699, 1234567891231709,
1234567891231739, 1234567891231811, 1234567891231837, 1234567891231853,
1234567891231903, 1234567891231949, 1234567891231967, 1234567891232057,
1234567891232059, 1234567891232069, 1234567891232107, 1234567891232287,
1234567891232317, 1234567891232387, 1234567891232407, 1234567891232471,
1234567891232537, 1234567891232581, 1234567891232587, 1234567891232671,
1234567891232713, 1234567891232759, 1234567891232779, 1234567891232789,
1234567891232803, 1234567891232933, 1234567891232939, 1234567891232993,
1234567891233007, 1234567891233029, 1234567891233031, 1234567891233043,
1234567891233091, 1234567891233113, 1234567891233119, 1234567891233169,
1234567891233191, 1234567891233217, 1234567891233241, 1234567891233263,
1234567891233269, 1234567891233319, 1234567891233323, 1234567891233401,
1234567891233427, 1234567891233431, 1234567891233473, 1234567891233529,
1234567891233539, 1234567891233577, 1234567891233581, 1234567891233619,
1234567891233623, 1234567891233671, 1234567891233709, 1234567891233731,
1234567891233827, 1234567891233829, 1234567891233841, 1234567891233877,
1234567891233953, 1234567891233959, 1234567891234009, 1234567891234037,
1234567891234081, 1234567891234103, 1234567891234169, 1234567891234193,

1234567891234211, 1234567891234213, 1234567891234319, 1234567891234333, 1234567891234343, 1234567891234369, 1234567891234397, 1234567891234409, 1234567891234451, 1234567891234459, 1234567891234477, 1234567891234499, 1234567891234529, 1234567891234537, 1234567891234541

• Total number of prime candidates in *output.txt*: 287

• The accuracy between prime candidates and primes in the online database: 100%

• Running time: 25.708 sec

4. Discussions

In general, it was proved that the prime wave analysis modified with a modular operator could be used as a practical technique for finding the new primes within a limited but extended number boundary. Using the known primes less than 3, for example, the new primes were determined within a boundary from 3^2 to 43046721^2. In the prime wave analysis (Eom 2024a), it could be explained that the known prime waves less than 3 were directly connected to composites; as a result, the composites were eliminated and the passively remaining numbers, which were not affected by the continued known prime waves, were all new primes within a limited but extended boundary from 3^2 to 43046721^2.

Through the process of finding the real primes, the roles of the known primes, the composites, and the passively remaining new primes were demonstrated, as visualized in Figure 2. Within n^2, a total of n boundaries could be generated from the 1st to the nth. The waves of known primes (filled circle) $sin\left(\frac{180}{p1}\cdot x\right)$ and $sin\,(\frac{180}{p2}\cdot x)$, where $p_2 < p_1 < n,$ rhythmically oscillated from the 1st to the nth boundary by eliminating the composites (filled star). The passively remaining primes could be visualized in an equation of $\frac{sin\,(180\cdot x)}{sin\left(\frac{180}{p1}\cdot x\right)\cdot sin(\frac{180}{p2}\cdot x)}$ within the limited boundary of n^2 (Eom 2024a). From the 2nd boundary, newly generated composites (unfilled star), such as $p_1{\cdot}p_2$, were produced, and their waves of intervals were larger than those of p_1 and p_2. As a result, this caused irregularities in the distribution of primes. For example, the waves of known primes, 2 and 3 in the 1st boundary eliminated the composites between the 2nd and 5th boundary. The prime number 5 was excluded because it was used as the boundary. Irregularities occurred due to the newly generated composites, such as 6, 12, 18, and 24, by the multiplication of 2 and 3. As a result, 7, 11, 13, 17, 19, and 23 remained passive and they were all new primes.

5. Conclusions

It was concluded that the prime wave analysis modified with a modular operator could be a practical technique for finding the new primes within a limited but extended boundary. Using the known primes less than 3, the new primes were found within a boundary from 3^2 to 6561^2 and it was extended to 6561^2 by finding the new primes within any selected boundary between 43046721 and 43046721^2. The results demonstrated the operational process of prime wave analysis: waves of known primes in the 1st boundary were directly connected to composites, while the passively remaining new primes were found between the 2nd and the *n*th last boundary. Therefore, the role of the 1st boundary was a key to understanding the prime wave analysis.

5. References

Eom, J. (2024a) Characteristics of primes within a limited number boundary. http://vixra.org/abs/2406.0046

Eom, J. (2024b) Approaching Goldbach’s conjecture using the asymmetric relationship between primes and composites within a limited number boundary. http://vixtra.org/abs/2406.0072

Guiasu, S. (2019) The proof of Goldbach’s conjecture on prime numbers. *Natural Science*. 11: 273-283. Doi.org/10.4236/ns.2019.119029

Stein W. (2000) Prime numbers. In *Elementary number theory: Primes, congruences, and secrets* (ed. S. Axler and K. A. Ribet). Springer, New York, pp 1-20. Doi 10.1007/978-0-387-85525-7

Figure 1. Schematic diagram of a modular operator designed based on prime wave analysis. The known primes listed in *input.txt* are used as factors for composites within a limited boundary. As a result, the composites are eliminated through the modular operator and the remaining numbers are all primes. The new primes are exported to *output.txt* and used for finding more primes in the extended number boundary.

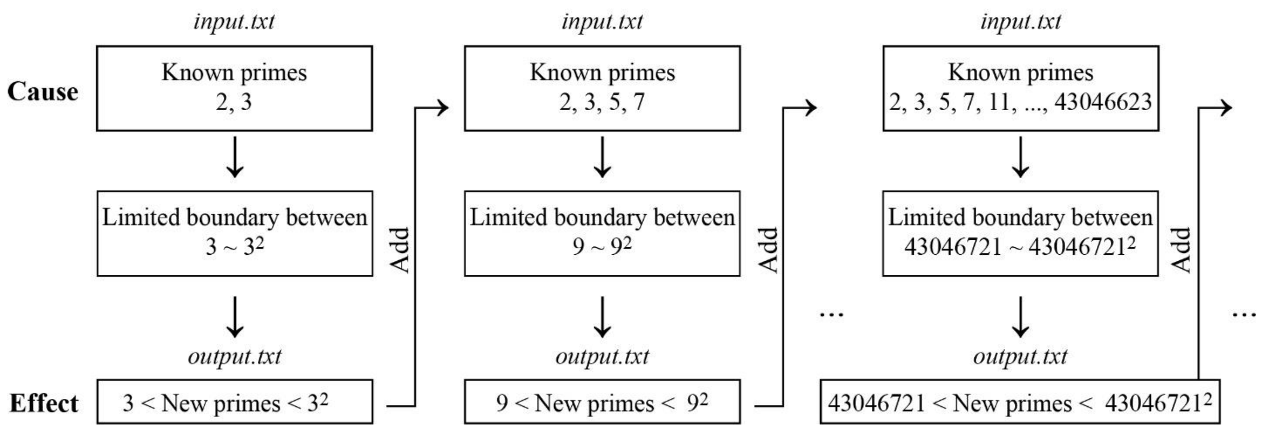
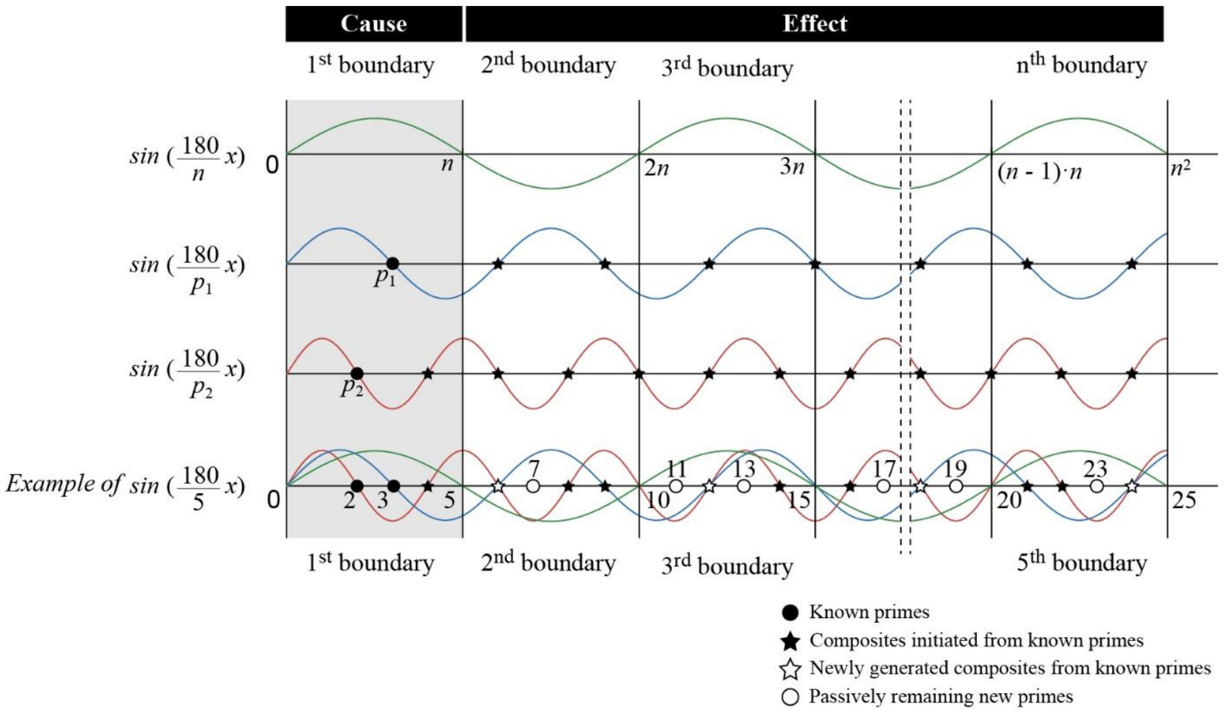

input.txt
Cause
Known primes 2, 3
Limited boundary between 3 ~ 3²
output.txt
Effect
3 < New primes < 3²
Add
input.txt
Known primes 2, 3, 5, 7
Limited boundary between 9 ~ 9²
output.txt
9 < New primes < 9²
Add
...
input.txt
Known primes 2, 3, 5, 7, 11, ..., 43046623
Limited boundary between 43046721 ~ 43046721²
output.txt
43046721 < New primes < 43046721²
Add
...

Figure 2. Analysis of Example 3 using the prime wave analysis. A) The waves of known primes (filled circle) in the 1st boundary are directly connected to composites (filled and open star); as a result, the composites are eliminated so the passively remaining numbers (open circle) are all new primes between the 2nd and nth boundary. Using the known primes less than 5, passively remaining new primes are found by eliminating the composites within a limited boundary between 5 and 5^2.



Cause
Effect
1st boundary
2nd boundary
3rd boundary
nth boundary
$sin(\frac{180}{n}x)$
$sin(\frac{180}{p_1}x)$
$sin(\frac{180}{p_2}x)$
Example of $sin(\frac{180}{5}x)$
0
n
$2n$
$3n$
$(n-1)\cdot n$
n^2
p_1
p_2
2
3
5
7
10
11
13
15
17
19
20
23
25
5th boundary
● Known primes
★ Composites initiated from known primes
☆ Newly generated composites from known primes
○ Passively remaining new primes