Prime number Research Program (PrP)

by Theo Kortekaas

Description and user guide PrP version 2.0

Purpose of the program is to generate and to manipulate prime numbers in the number domain from 0 to 2^{64}. The program is designed to operate in a 32-bit Windows environment on a typical desktop or laptop computer with a main memory of at least one gigabyte.
Manipulation can be defined as: count primes; generate twin primes; count prime gaps and manufacture statistics about prime gaps; search for prime k-tuple constellations.
The method used to generate prime numbers is the sieve of Eratosthenes. See website:
en.wikipedia.org/wiki/Sieve_of_Eratosthenes

The sieve of Eratosthenes

To generate prime numbers up to 2^{64} with the sieve of Eratosthenes you need prime numbers up to 2^{32} (root 2^{64} is 2^{32}). Furthermore, a sequence of consecutive numbers is required from 0 to 2^{64}. We will see that this is not possible (with the currently available computers) and what alternative possibilities there are.

The first action of the PrP program is to set up a table with all the prime numbers up to 2^{32}. (This is also done using the sieve of Eratosthenes). That are 203,280,221 prime numbers. These primes can be up to 32 bits long. So four bytes are required to store each prime. A complete table would cover more than 800 megabytes. Because this table is to be used frequently, it should be in computer memory permanently. That puts too much pressure on the computer memory.
It was decided not to store prime numbers itself, but to store the gaps between primes in a table. The largest primegap that can occur between prime numbers up to 2^{32} is 320. As all prime gaps (with one exception) form an even number, the value of a primegap, divided by two, can be stored. The greatest value to be stored is then 320/2 = 160. This value can be contained in one byte. When the prime number is regenerated, the stored value must then be multiplied again by two.
A disadvantage of this method is that the prime numbers can only be retrieved by going through the table sequently from the beginning. This is for the use of

the table in the sieve of Eratosthenes no objection.
The gap between the first two primes (two and three) is not even. So this gapvalue cannot be divided by two. Therefore this two primenumbers are stored in the table fully, each in one byte. The index for the elements in the table starts at zero. It is decided to synchronize the index of the table with the sequence number of the prime. So the first element (byte) of the table with index 0 gets the content zero; the second byte gets the content 2 (index 1 and the first prime is 2); the third byte gets 3 as value (index 2 and the second prime is 3). The first part of the table now looks like:

0 First byte; intentional zero for the index is zero.
2 (first prime number)
3 (second prime number)
1 (*2 is primegap between 3 and 5)
1 (*2 is primegap between 5 and 7)
2 (*2 is primegap between 7 and 11)
1 (*2 is primegap between 11 and 13)
2 (*2 is primegap between 13 and 17)

So this are the first eight bytes for the first seven prime numbers. In total the table will contain 203,280,222 bytes for all prime numbers less than 2^{32}. This table has to fit in computer memory; a copy of the table must reside on disc.

Additionally a row of sequential numbers is required in computer memory; the numbers in this row must can be marked to be composite. The smallest possible indicator in computer memory is a bit. However a bit is not easily separately addressable; additional computer instructions are needed to manipulate bits. From a viewpoint of computer memory space, a bit is the optimum choice. However for reasons of speed it may be better to use a byte. A byte can be individually addressed and can simple be put on or off by means of one computer instruction.
In the ideal situation a row of numbers from 0 to 2^{64}, represented by 2^{64} bytes, should be used in computer memory. However the practical limit of current desktop or laptop computers (running a 32-bit Windows) is about one billion bytes free memory. So to work with numbers in the range of up to 2^{64} only piece by piece can be treated at the same time, in pieces of maximum one billion. This pieces are called segments; the sieve of Eratosthenes is segmented.

Search space

The program provides the ability to specify a sequence of numbers. This seqence is called “number range” and is specified by a “from”number and a “to” number. The number range forms the *search space* where the program looks for prime numbers.
The *search space* can start at zero but that is not necessarily required. The search space may be defined as large as is desirable, but the larger the search space the more time it costs to process it.

The segmented sieve

The original design of the sieve of Eratosthenes is that prime numbers from the begin of the number array are used to mark out multiples (as composite numbers) of these primes later on in the number array. The numbers that remain unmarked forms at the end the prime numbers in the number array.
This only works if the number array starts at zero.
Because the PrP program has available a table with all primes up to 2^{32} there is no need to start the number array (*search space*) with zero.
If the specified *search space* is too large for the computer memory, then the *search space* can be segmented. The size of a segment can be specified.

For part of the number range, space is reserved in computer memory of the size of a segment, one byte per number. Operations are executed on this segment. Then for the next segment the computer memory is prepared and operations are executed and so on. This continues until the entire search space is handled. The result of the operations are reported for each segment and a total for the entire search space.

User Guide

The Prime Number Research Program can be downloaded from
www.tonjanee.home.xs4all.nl/downle.html
After downloading, the file extension must be changed from .pgm to .exe and the program must be transferred to a map, preferably a separate map for primes (for instance a map with the name "prime"). In the rest of this guide this map will be referred to as *"the map"*.
The map should be located on a drive where there is a minimum of about 250 megabytes of free space. After the program has started the first messages of the program are displayed.

The display

On the computer monitor the program creates two display areas. The upper display area is used to enter data; the lower area is a "scroll area". In the scroll area messages and instructions of the program are displayed and the results of the calculations. Each new line is added to the bottom af the scroll area; the exsisting lines are moved up one place and the upper line disappears when it comes out of the scroll area.

Online log and logfile

The lines that are displayed in the scroll area are also placed in an online log. This log can accomodate about 12,000 lines. When the log is full, new lines are placed in the begin of the log, overwriting old lines. This is called "wrap-around". If this occurs a message is placed in the log. All the data from the online log is also placed in a log file in *the map*. For this there must be sufficient free space available in *the map*.

After starting the program it is possible to change the language (currently only English and Dutch are available).

Reading the primegap file

The first action must be to read in the primegap file named "PGTABLE.TAB" or to create this table when it is not present in *the map.* This is initiated by selecting "Set Up" in the main menu and subsequently selecting "Read in or build the primegap table". If the primegap file already exists in *the map*, then this file is read. That can take a couple of seconds. When a problem occurs when reading the file the program returns an error message and stops. In that

case the primegap file must be removed manually. Then the program can be started again.
If no problem occurs the program returns the message “Filegap file read successfully”. If no primegap file exists in the folder then the file is created. This may take several minutes. During the creation progress is reported.

Specification of number range

After the primegap file is created or succesfully read, the number range of the *search space* can be specified. This done by selecting “SetUp” in the main menu and subsequently selecting “Define begin and end of number range” An image of a numeric keypad is displayed. On this keypad a number can be entered with the mouse.
This is the begin number of the number range. The number should be a minimum of zero and a maximum of 18,446,744,073,709,550,000. The number must not be odd. If the number is keyed in correctly then the OK button is pressed.
Hereafter the end number is entered the same way as the begin number. This number must be greater than the begin number, and must be at least 16 and may reach a maximum of 18,446,744,073,709,551,616 (this is 2^{64}). Again this number must be even (not odd) and must be concluded with the OK button.

Now the size of a segment can be entered. The number that is entered specifies the amount of “memibytes” that is used as sieve area. (One “memibyte” , symbol MiB, is 1024 x 1024 bytes = 1,048,576 bytes. The expression “memibyte” was introduced in 1998 by the International Electrotechnical Commision as a new standard). The size of a segment is minimum 1 MiB and maximum 1,024 MiB. The number must be concluded with the OK button.

Remark:
The difference between the begin number and the end number represents the *search space*. The size of the *search space* determines the duration of the various actions that can be performed. Depending on computer speed, memory size and segment size, the program can evaluate from 10 million to 60 milllion numbers per second on being prime. For a *search space* of one trillion (10^{12}) a processing time of a couple of hours to more than one day can be expected. The segment size is an important factor in the performance of the program. Varying the segment size one can find the optimum for a given computer environment.

Actions

There are seven different actions that can be performed on the *search space.* These actions represent the real thing.
Once a *search space* has been defined, one or more actions in any order may be performed consecutively on the same *search space*. An action is initiated by selecting “Action” in the main menu and subsequently selecting one of seven different actions. Results are reported per segment and totals for the entire search space. This applies for every type of action. Results are displayed in the scroll-area.

Action 1. Count primes and twin primes

All prime numbers in the search space are counted as well as all twin primes and the results are reported by segment. A twin prime is a couple of prime numbers that differ only two. For example 11 and 13 form a twin prime as well as 17 and 19.Twin primes are also counted in number of prime numbers. Every twin prime is counted for two in the twin primes count. The prime numbers 3, 5 and 7 form a prime triplet; therefore they are not counted in number of twin primes.

Note:
If a twin prime exists in the beginning of the search space, but the smallest prime of the twin does not fall within the search space boundary, this twin is not counted in number of twin primes. The same applies if a twin prime exists at the end of the search space but the largest prime of the twin does not fall within the boundary of the search space.

Action 2. Calculate and display Prime Numbers

All prime numbers that fall in the search space are shown in the scroll-area of the display, one line at a prime number. This happens so fast that individual primes are barely distinguisable. This action can only be meaningful if the search space is limited and the results, written in the logfile, are used.

Action 3. Calculate and display Twin Primes

All twin primes that exist within the *search space* are shown in the scroll-area of the display, two lines per twin prime with one separation line. This happens so fast that individual twin primes are barely distinguisable. This action can

only be meaningful if the search space is limited and the results, written in the logfile, are used. When the s*earch space* starts with zero, then the first twin prime shown is 11 and 13. Prime numbers 3, 5 and 7 are not counted as twin prime.

Action 4. Calculate and display Prime gaps

The gaps between all primes within the *search space* are calculated and these gaps are counted by gap size. By segment the counts are reported and at the end of the *search space* totals are reported for the whole *search space* by gap size. Additional for each gap size the prime number is reported at which the gap size was found for the first time. This is valid only for the prime numbers within the *search space*. Prime gaps at the beginning of the s*earch space* and at the end of the *search space* are not included in the count.

Action 5. Show Most Lonely Prime Numbers

As prime numbers get bigger they are further apart. But the prime numbers are rather randomly distributed over all the numbers. Sometimes primes are close together, sometimes there is a big gap between two consecutive primes. How lonely primes can be?
We first define loneliness for primes. Suppose we have a prime number x. The prime prior to x we call w; the prime next to x is called y. The distance from x to the prior prime is $(x-w)$; the distance from x to the next prime is $(y-x)$. We now take as measure of loneliness the smallest value of $(x-w)$ and $(y-x)$. The larger this value is the greater is the degree of loneliness. There are other definitions possible but for the PrP program we adhere to the definition described here.
For the first prime in the search space we set the degree of loneliness to zero. For the second prime we can determine the degree of loneliness as described above. This prime is than the most lonely prime sofar. For every following prime we determine the degree of loneliness and if this is more than de previous most lonely prime, it becomes the next most lonely prime.
If the search space does not begin at zero, than a prime number found loneliest is not absolutely the loneliest, but the loneliest in the colllection of primes in te search space.
In the first segment of a segmented search space some loneliest primes are found quickly; but in subsequent segments no primes may be found lonelier than the loneliest in the previous segment. In that case only the number of primes in that segment are reported.

Action 6. Show Most Lonely Twin Primes

Prime twins are never lonely because they have each other. But we define loneliness for a twin prime as the smallest distance of the twin prime to the first previous prime or the first next prime.

Prime twins are never lonely because they have each other. But we define loneliness for a twin prime as the smallest distance of the twin prime to the first previous prime or the first next prime. Suppose we have a twin prime existing of a first prime number x and the second prime number $x+2$.
The prime prior to x we call w; the prime next to $x+2$ is called y. The distance from x to the prior prime is $(x-w)$; the distance from $x+2$ to the next prime is $(y-x-2)$. We now take as measure of loneliness the smallest value of $(x-w)$ and $(y-x-2)$. The larger this value is the greater is the degree of loneliness. There are other definitions possible but for the PrP program we adhere to the definition described here.
If the search space starts at zero, then we can for the first twin prime determine the degree of loneliness. This is then until the next twin prime the loneliest twin prime. For each next twin prime we also determine the degree of loneliness and when the degree of loneliness is greater than that of the previous loneliest twin prime, then the new twin prime becomes the next loneliest twin prime.
If the search space does not start at zero then the first encountered twin prime is not in absolute terms the loniest twin prime but the loneliest twin prime since the start of the search space.

Action 7. Calculate and display Palprimes (deleted); replaced by:

Action 7. Prime k-tuple clusters (constellations) search

A prime k-tuple is a pattern of consecutive primes. The pattern is defined by k numbers representing the intervals between a base prime number p and the successive prime numbers. The pattern is depicted as $(n_1, n_2, n_3, \ldots n_k)$. Herein is the first term, n_1 usually 0, and represents the basenumber p (p+n_1). The following terms represent the interval between the base p and the next prime numbers. So the base prime number = p; the following primes are: p+n_2; p+n_3; ... p+n_k. The objective is to search for base primenumbers p such that p+n_2, p+n_3 ... p+n_k are all primes. For example, k-tuple (0,2) forms the pattern for twin primes.
Because all prime numbers (except 2) are odd numbers, the intervals between two odd prime numbers form always an even number. So n_1, n_2, .. n_k must be

even numbers. Further, the terms of the k-tuple must be specified in order of ascending values.
Not all values form a "valid" prime k-tuple. For "valid" prime k-tuples the "*prime k-tuple conjecture"* applies, that there are infinitely many base primes *p* wherein the (k - 1) following numbers also are all primes. These k-tuples are called *admissible*. The set of k consecutive prime numbers that conforms to the pattern of a prime k-tuple is referred to as a "prime cluster" in this document as well as in the PrP program. Also the term "prime constellation" is used, but the meaning of "prime constellation" may vary slightly, depending on the article or document, in which the term is used.
The "not valid" k-tuples are called *inadmissible*. These tuples may have none, one or only a limited valid set of primes.
For example, the prime k-tuple (0,2,4) is an inadmissible k-tuple, because there are no sets of three prime numbers that differ two from each other, with the exception of (3,5,7). With larger prime numbers, if the first prime number of the set, by division by three, gives a remainder of 1; then necessarily the second is divisible by three. If the first primenumber of the set, by division by three, gives a remainder of 2, then the third prime number is necessarily divisible by 3.

After the search space is set and action 7 is selected, a new window is displayed, in which the prime k-tuple can be entered. The first number is zero, and is filled in automatically. Then numbers are entered in order of value, separated by a comma. (The C button is used as the comma key). The maximum value of a number is 98. Up to ten numbers can be entered; all numbers should be even.
When the prime k-tuple is entered correctly the OK button can be pressed. The search process will then start. If clusters are found they are displayed as k+1 lines: the first line displays the k-tuple, the following line shows the beginning prime number (to be referred to as Base prime number). Then follow the next prime numbers; one per line.
All found clusters in the *search space* are shown. Clusters that begin before the beginning of the *search space* are not shown; Also clusters ending after the end of the *search space* are not displayed. If there are two (or more) clusters that overlap one another, then both (or all) clusters are shown and both or all are counted as a prime cluster. An example of overlapping clusters is k-tuple (0,6,12) for base prime number $p = 251$ (251,257,263), and $p = 257$ (257,263,269).
The number of constellations per segment is reported, as well as the total of the entire *search space*.
See: https://en.wikipedia.org/wiki/Prime_k-tuple

The Log File

At the start of the program a log file is opened on disk.
This file will get a name that contains the date and time when the program is started as follows: DDMMUUmm.LOG.
DD is the day and MM is the month; HH is the hour and mm is the minute of the moment the program is started. If the program is started twice in succession immediately one after another, there is a risk that the second log would be given the same name as the first log file. In that case, the creation of the second file failes. No mention is made in the log.