Discrete Hyperbolic Polylogarithmic Sieve For Integer Factorization - using Interpolation Search (Draft)

Ka.Shrinivaasan (ka.shrinivaasan@gmail.com)

June 30, 2013

Abstract

Best known algorithm for factorization is the General number field sieve algorithm which takes time exponential in number bits in the input or simply called exponential algorithm. In this article, a polylogarithmic time (in size of input) sieve for factorization is discussed that uses elementary principles.

1 Discretization of a hyperbolic arc

1. A plot of a hyperbolic function is ($xy = N$) drawn on a 2-dimensional plane. This plot of hyperbola is continuous. This hyperbola can be discretized (or tesselated) with set of rectangles that cover the hyperbolic arc.

2. Discretization step divides the x-axis of the hyperbola into $log^c N$ intervals where each interval is of length $N/log^c N$. Infact it suffices to calculate upto $\sqrt{N}$ length along the x-axis since atleast 1 factor must lie within $\sqrt{N}$.Correspondingly the y-axis is also split into intervals whose width is $y * \sqrt{N}/(xlog^c N + \sqrt{N})$ where (x, y) are coordinates of topleft corners of each rectangle.

3. The constant c is chosen to be quite large so as to shrink the size of each rectangle as much as possible.Thus each rectangle is of length $\sqrt{N}/log^c N$ and width $y * \sqrt{N}/(xlog^c N + \sqrt{N})$ where (x, y) are coordinates of topleft corners of each rectangle. The factors (say p,q) of N are sure to be present within one of the rectangles if N is composite. Algorithm then is to sieve out rectangles to get to the factors. Looping through $log^c N$ rectangles takes time $O(log^c N * TimePerRectangle)$.

4. Within each rectangle,the values of coordinates are hypothetically considered to be products of x and y coordinates. This is just supposed to exist and no lookup table or memoization is needed (though it can be made into a dynamic programming recurrence algorithm if knowledge of factors of numbers lower than N is assumed). To find the coordinates where N is located within the rectangle, Interpolation search, a variant of binary search that takes $loglogN$ time on the average, is applied on the coordinate products with some specialized modifications as described in next section below.

5. The line equation corresponding to leading diagonal within each rectangle obtained previously, intersects with the hyperbola,thereby creating a bow-shaped area (hemmed between hyperbolic arc and diagonal) of coordinate products within each rectangle.This bow shaped area is only a small percentage of the area of each rectangle.

6. The row slices of coordinate products within this bow shaped area are in sorted order from left to right. Also the coordinate products in diagonal of the rectangle from bottom-left to top-right of each rectangle are in sorted ascending order. Thus searching these row slices of coordinate products (with some constant extension to each slice on leftside) within each rectangle is sufficient to sieve out the factors.

2 Interpolation search on the above bow-shaped area within each rectangle

1. The hyperbolic bow arc area obtained above can be searched with interpolation search on each of the row slice within a rectangle (along the y-axis).

2.
```
Functions InterpolationSearchSubroutine(xleft, yleft, xright, yright)
{
if(yleft == yright) //horizontal scan on a row slice yleft==yright
{
if (xleft+xright)/2 * yleft == N
{
return (xleft+xright)/2, yleft as factors of N
}
else if (xleft + xright)/2 * yleft > N
InterpolationSearchSubroutine(xleft, yleft, (xleft+xright)/2, yright)
else
InterpolationSearchSubroutine((xleft+xright)/2, yleft, xright, yright)
}
}
```

3 Discrete Hyperbolic Polylogarithmic Sieve for Integer Factorization - Algorithm Steps

1. Before the factorization loop starts for a number N to be factorized, the constant c has to be chosen suitably. This constant decides the parallelism as described later.

2. Split interval $[0 - N]$ into $log^c N$ intervals of width $N/log^c N$ each.

3. For each of $log^c N$ rectangle of width $N/log^c N$ and height $y * \sqrt{N}/(xlog^c N + \sqrt{N})$ where (x, y) is the coordinate of top-left corner of each rectangle, compute the following loop:

   (a) By Interpolation Search, using InterpolationSearchSubroutine described above, search each row slice within the bow shaped area confined within the hyperbolic area and the diagonal of the rectangle.

   (b) Output $p, q(pq = N)$ if found in previous step and break; else continue loop

Low level calculations are derived later and also described in attached handwritten notes in appendix. This algorithm takes average upperbound time of $O(log^c N * loglogN)$.

4 Implications

As a consequence of this polylog algorithm for factorization, RSA cryptosystem's large prime keypair can be found in polynomial time by factorization for which till now there are only BQP quantum polynomial algorithms available. Hence factorization in P implies probably that $BQP = P$ (if factorization is BQP-complete) since there is only a quantum polynomial algorithm for factorization and implies therefore that $BPP = P$ since BQP contains BPP (implying pseudorandomness and determinism are equivalent in computational power). This algorithm outputs all factors of an integer within polylogarithmic time in input bits and as a special case this is also a polynomial time primality test similar to PRIMES in P (AKS).

5 Derivation of the upperbound time for discrete hyperbolic factorization

Theorem 1. *Discrete Hyperbolic Polylogarithmic Factorization has an average upperbound running time of* $O(log^{2c+2}N)$

Proof. Area of hyperbolic bow area per rectangle = Area of rectangle - Area of hyperbolic arc(obtained by area definite integral)

$=(1/2 * N/log^cN * y_1\sqrt{N})/(2 * log^cN * (x_1log^c + sqrt(N))) - Nlog(x_1 + N/log^cN) + Nlogx_1$

Dividing above bow area by y-axis side of the rectangle $deltay$ gives average length of each hyperbolic row slice within bow: $((1/2 * N/log^cN * y_1\sqrt{N})/(2 * log^cN * (x_1log^c + sqrt(N))) - Nlog(x_1 + N/log^cN) + Nlogx_1)/(y_1 * \sqrt{(}N)/(x_1 * (log^cN) + \sqrt{(}N)))$

The above is average length of each hyperbolic row slice within bow area that needs to be interpolation searched. Time taken for interpolation search of row slice that is in ascending order from left to right will be $loglog(aboveexpression)$:

$loglog(((1/2 * N/log^cN * y_1\sqrt{N})/(2 * log^cN * (x_1log^c + sqrt(N))) - Nlog(x_1 + N/log^cN) + Nlogx_1)/(y_1 * \sqrt{(}N)/(x_1 * (log^cN) + \sqrt{(}N))))$

Above term can be upperbounded as follows ignoring some subtraction and division terms:

$Aboveexpression \leq loglog(((1/2*N/log^cN*y_1\sqrt{N})/(2*log^cN*(x_1log^c+sqrt(N)))+Nlogx_1)/(y_1* \sqrt{(}N)/(x_1 * (log^cN) + \sqrt{(}N))))$

$Aboveexpression \leq loglog((N * y_1\sqrt{N}) + Nlogx_1) * 2 * log^cN * (x_1log^cN + \sqrt{(}N))/(2 * log^cN * y_1 * \sqrt{(}N))))$

$Aboveexpression \leq loglog((N/2log^cN) + (Nlogx_1(x_1log^cN + \sqrt{(}N)))/(y_1 * \sqrt{(}N))$ Since $y_1 = N/x_1$, $Aboveexpression \leq loglog((N/2log^cN) + (x_1 * logx_1(x_1log^cN + \sqrt{(}N)))/(\sqrt{(}N))$

x_1 can take $\sqrt{N}$ as maximum (if only $\sqrt{(}N)$ fraction of x-axis is scanned with rectangular sieves):

$Aboveexpression \leq loglog((N/2log^cN) + (log\sqrt{N}(\sqrt{N}log^cN + \sqrt{(}N))))$

$Aboveexpression \leq loglog(N + (\sqrt{N}/2)(log^{c+1}N + log(N)))$

$Aboveexpression \leq loglog(2 * N * log^{c+1}N)$

$Aboveexpression \leq loglog(2 * N * log^{c+1}N)$ or $loglog(2 * N(c+1) * logN)$

Thus time per row slice interpolation search per rectangle is upperbounded by:

$loglog(q * N * logN)$ *for* $q = 2c + 2$

Multiplying above upperbound for $deltay_i$ rows within each $rectangle_i$ and for all log^cN rectangles, gives the total running time:

$\sum_{N/log^cN}^{N} N\sqrt{(}N)/(x_1(x_1log^cN + \sqrt{(}N)))$ with $deltax = N/log^cN$

Ignoring $x_i = 1$ consecutive $x_i(s)$ are: (each rectangle is N/log^cN wide)

N/log^cN,$2N/log^cN$,$3N/log^cN$,...,$(log^c)N/log^cN$

Above summation can be reduced as:
$N\sqrt(N) * [(1/(N/log^cN(N/log^cN * log^cN + \sqrt(N))) + ...]$
$N\sqrt(N) * log^cN/N * (1/(N + \sqrt(N)) + 1/(2(2N + \sqrt(N)) + ... + 1/(log^cN(log^cN * N + \sqrt(N)))$
$aboveexpression \leq log^cN(1/\sqrt(N) + 1/(2^2\sqrt(N)) + 1/(3^2\sqrt(N)) + ... + 1/(log^{2c}N\sqrt(N)))$
$aboveexpression \leq log^cN(1 + 1/(2^2) + 1/(3^2) + ... + 1/(log^{2c}N))$
$aboveexpression \leq (log^cN/\sqrt(N)) * RiemannZetaFunction(s = 2)$
$aboveexpression \leq (log^cN/\sqrt(N)) * (\pi^2/6)$
Thus total time for interpolation search of all row slices within all rectangles is : $O((log^cN/\sqrt(N)) * 3loglog(q * N * logN)$ or simply $O((log^cN * loglog(N))$

In the calculations above the part of x-axis only upto $\sqrt(N)$ is divided into log^cN intervals of width. The calculations for dividing x-axis upto N are below:

1. deltax = N/log^cN and deltay = $N^2/(x_i(x_ilog^cN+N))$ where x_i(s) are N/log^cN,$2N/log^cN$,$3N/log^cN$,...,(log^c)

   Area of hyperbolic bow area per rectangle = Area of rectangle - Area of hyperbolic arc(obtained by area definite integral)

   $=(1/2 * N/log^cN * y_1 * N)/(2 * log^cN * (x_1log^c + N)) - Nlog(x_1 + N/log^cN) + Nlogx_1$

   Dividing above bow area by y-axis side of the rectangle *deltay* gives average length of each hyperbolic row slice within bow: $((1/2 * N/log^cN * y_1 * N)/(2 * log^cN * (x_1log^c + N)) - Nlog(x_1 + N/log^cN) + Nlogx_1)/(y_1 * N/(x_1 * (log^cN) + N))$

   The above is average length of each hyperbolic row slice within bow area that needs to be interpolation searched. Time taken for interpolation search of row slice that is in ascending order from left to right will be $loglog(aboveexpression)$:

   $loglog(((1/2*N/log^cN*y_1*N)/(2*log^cN*(x_1log^c+N))-Nlog(x_1+N/log^cN)+Nlogx_1)/(y_1*N/(x_1*(log^cN)+N)))$

   Above term can be upperbounded as follows ignoring some subtraction and division terms:

   $Aboveexpression \leq loglog(((1/2 * N/log^cN * y_1N)/(2 * log^cN * (x_1log^c + N)) + Nlogx_1)/(y_1 * N/(x_1 * (log^cN) + N)))$

   $Aboveexpression \leq loglog((N*y_1*N)+Nlogx_1)*2*log^cN*(x_1log^cN+N)/(2*log^cN*y_1*N)))$

   $Aboveexpression \leq loglog((N/2log^cN) + (Nlogx_1(x_1log^cN + N))/(y_1 * N)$ Since $y_1 = N/x_1$,
   $Aboveexpression \leq loglog((N/2log^cN) + (x_1 * logx_1(x_1log^cN + N))/(N)$

   x_1 can take $\sqrt{N}$ as maximum (if only $\sqrt(N)$ fraction of x-axis is scanned with rectangular sieves):

   $Aboveexpression \leq loglog((N/2log^cN) + (logN(Nlog^cN + N)))$

   $Aboveexpression \leq loglog(N + (N/2)(log^{c+1}N + log(N)))$

   $Aboveexpression \leq loglog(2 * N * log^{c+1}N)$

   $Aboveexpression \leq loglog(2 * N * log^{c+1}N)$ or $loglog(2 * N(c + 1) * logN)$

   Thus time per row slice interpolation search per rectangle is upperbounded by:

   $loglog(q * N * logN)$ *for* $q = 2c + 2$

   Multiplying above upperbound for $deltay_i$ rows within each $rectangle_i$ and for all log^cN rectangles, gives the total running time:

   $\sum_{N/log^cN}^{N} N * N/(x_1(x_1log^cN + N))$ with $deltax = N/log^cN$

   Ignoring $x_i = 1$ consecutive $x_i(s)$ are: (each rectangle is N/log^cN wide)

N/log^cN,$2N/log^cN$,$3N/log^cN$,...,$(log^c)N/log^cN$

Above summation can be reduced as:

$N * N * [(1/(N/log^cN(N/log^cN * log^cN + N)) + ...]$

$N * N * log^cN/N * (1/(N+N) + 1/(2(2N+N) + ... + 1/(log^cN(log^cN * N + N))$

$aboveexpression \leq log^cN(1/(N) + 1/(2^2 * (N)) + 1/(3^2 * (N)) + ... + 1/(log^{2c}N * (N)))$

$aboveexpression \leq log^cN(1 + 1/(2^2) + 1/(3^2) + ... + 1/(log^{2c}N))$

$aboveexpression \leq (log^cN/) * RiemannZetaFunction(s = 2)$

$aboveexpression \leq (log^cN) * (\pi^2/6)$

Thus total time for interpolation search of all row slices within all rectangles (from N/log_cN to N) is : $O((log^cN/\sqrt(N)) * 3loglog(q * N * logN)$ or simply $O((log^cN * loglog(N))$

Above has excluded the strip of x-axis from 1 to N/log^cN. This is just because it suffices to start the tesselation from some point on x-axis that is less than $\sqrt(N)$ rather than from 1 and upto N. Also it helps in getting to Riemann Zeta Function with $s = 2$ to upperbound the summation. With this atleast 1 factor can be sieved out and to extract all factors above algorithm is recursively applied by dividing with already found factors. This process is only polynomial in logN. Such a point can be found by suitably approximating the square root (actual square root finding is factorization-hard) and finding c. In other words:

Find c such that, $\sqrt(N) > N/log^cN$ or $log^cN > \sqrt(N)$ which reduces to: $c > logN/2loglogN$.Before starting factorization such a constant c can be found satisfying above inequality.

If approximate square root finding is not preferable, above algorithm can be recursively applied to the strip of x-axis from 1 to N/log^cN. This would give a series summation of upperbounds as: $S = O(log^cN) + O(log^c(N/log^cN)) + O(log^c(N/log^{2c}N)) + ... + O(log^c(N/log^{qc}N))$

$S = O(log(N) + log(N/log^cN) + ... + log(N/log^{qc}N))^{c+1}$

$S = O((q+1)log(N))^{c+1}$

The above recursion stops when $log(N/log^{qc}N) = log1 = 0$.

Solving for q:

$N = log^{qc}N$

$log(N) = qc(loglog(N))$

$q = logN/(cloglogN)$. Thus recursion stops after q iterations.

$S = O((logN/cloglogN + 1)logN)^{c+1}$

Hence total running time is $O(((logN/cloglogN + 1)^{c+1}(logN)^{c+1})$ or $O(log^{2c+2}N)$ with no restriction on choice of constant c. Value for c gives some flexibility in deciding the number of rectangles that tesselate the hyperbola.If we have log^cN number of processors which would imply an NC circuit, above factorization loop can infact be done using a logdepth, bounded width NC1 circuit with each rectangle assigned to one processor (if interpolation search per bow arc area within each rectangle is also in logdepth) and constant c decides the extent of parallelism.

Thus above series obviously converges with only polylog recursive iterations and thus total time taken is still polylog in input size.This proves the correctness of the polylog upperbound for Discrete Hyperbolic Polylogarithmic Sieve For Integer Factorization.

□

6 Acknowledgement

I dedicate this article to God.

7 Bibliography

References

[1] Elementary algebraic and geometric principles

[2] Binary Search, Interpolation Search

[3] Riemann Zeta Function