ANALYZING THE QUADRATIC SIEVE ALGORITHM

KIM BOWMAN, NEIL CALKIN, ZACH COCHRAN, AND KEVIN JAMES

Abstract. We will discuss random vector models related to the quadratic sieve. We will introduce a new model for binary vectors, suggesting that we can cut the running time of the quadratic sieve. Also, we will prove an upper bound for the number of vectors needed to create a dependency.

1. Introduction

Over the years, mathematicians have used several methods to try to factor large numbers. Originally, one of the only known ways to factor was to do so by trial division. Fermat came up with the technique of writing a number as the difference between two squares, i.e. $n = a^2 - b^2$. This works since:

$$n = a^2 - b^2 = (a + b)(a - b)$$

Therefore, $(a + b)$ and $(a - b)$ are factors of n. Fermat's method is a step, but its still a lengthy process. Kraitchik improved this method later by saying that n did not need to equal $a^2 - b^2$, but it was sufficient enough to say that $a^2 \equiv b^2 \bmod n$, meaning $a^2 - b^2 = tn$ for some integer t. Now, compute the GCD$(n, a - b)$. This may not always result in a factor, but the probability for finding one is fairly high. In search for the a's and b's, Kraitchik said to find squares that were greater than n and compute them in an equation $a^2 - n$. [3]

After Kraitchik devised this idea, Dixon took it a step further again. Dixon's idea was to take several r_i values, where $r_i > \sqrt{n}$, and input them in the equation $g(r) = r^2 \bmod n$. Next, one would take the $g(r_i)$ values and attempt to completely factor these by trial dividing for primes up to some bound B. Then, Dixon said to find a set, $\{R_1, R_2, \ldots, R_m\}$, of various $g(r_i)$'s such that the exponents of the primes in the factorization of the product of these $g(r_i)$'s are all even. This means that $\prod_{i=1}^{m} g(R_i)$ is a square.

One of the easiest ways to find this combination is to put each $g(r_i)$'s exponential powers modulo two in a binary matrix where each column represents a different prime and each row is a different $g(r_i)$. This gives you a matrix of zeros and ones that is fairly sparse, meaning there are

few ones. To find the aforementioned set, one can perform Gaussian elimination on the newly formed matrix. The essence of *Gaussian elimination* is to perform elementary row operations on the matrix until a row of all zeros exists. This indicates that a linear dependency exists in the matrix. By knowing this, we know that we can find a product of $g(r)$'s where all the powers of the primes are even. So, we know that there's a set of R_n's such that the product of $g(R_i)$'s is a square modulo n. We can then let $a = \prod_{i=1}^{m} R_i$ and $b^2 = \prod_{i=1}^{m} g(R_i)$. Then, we know that $a^2 \equiv b^2 \bmod n$. So, going back to Fermat and Kraitchik, we can try to compute the $GCD(n, a - b)$ and see if we get a factor.

2. Quadratic Sieve

Created in 1981 by Carl Pomerance[2], the *quadratic sieve*, or QS, is an algorithm to factor numbers that extends the ideas of Kraitchik and Dixon. One way this sieve improves their ideas is by choosing the r_i values to take. In the QS, one takes an interval of r_i values between $\sqrt{n}$ and $\sqrt{2n}$. Also, instead of the aforementioned $g(r)$ function, the QS uses $f(r_i) = r_i^2 - n$ and places them in an array as in the one shown below.

$f(r_1)$	$f(r_2)$	$f(r_3)$	...	$f(r_n)$

Now, as before, this algorithm starts with a set of primes less than some bound B. However, one needs to compute the Legendre symbol $(\frac{n}{p_i})$for each p_i, where n is the number to be factored. The *Legendre symbol*, $\left(\frac{n}{p}\right)$ is:

$$\begin{cases} 1 & \text{if } a \text{ is a square mod } p \\ -1 & \text{if } a \text{ is not a square mod } p \\ 0 & \text{if } p \mid n \end{cases}$$

The primes that have a Legendre symbol equal to one are kept and the others are discarded. This reduced set of primes is known as the factor base. We use these primes because primes outside our factor base will not divide $f(r)$. The reason for this is that given any prime p, in order for p to divide $r^2 - n$, n musct be a quadratic residue modulo p. This is the exact condition that the Legendre symbol, $(\frac{n}{p}) = 1$.

The next step is called the sieving process and involves dividing each entry in the array by each prime as many times as possible. After doing this for every prime in the factor base, we find all the entries $A(j)$, $j = f(r_i)$ for some r_i, such that $A(j) = 1$. This set becomes the $(R_1, R_2, R_3, \ldots, R_m)$ set discussed earlier. The prime factorizations of

these entries are recorded, and the powers of the exponents of this set are put into the binary matrix for Gaussian elimination.

As stands, the QS is a time consuming algorithm. The lengthiest part of the algorithm is the sieving process itself, especially for the small primes used. The Gaussian elimination process is also a lengthy process. If there are k primes in our factor base, then there will be k columns in our matrix. This means we will need $k+1$ vectors, or $k+1$ $f(r_i)$ values, to find a dependency. This is because having more rows than columns in the matrix will guarantee a dependency.

So, if the problem is the time that it takes, then the question that arises is if the algorithm can be shortened in any way. The idea is to view the problem in a probabilistic manner. To go about this, we can look at the probability that a dependency is found in a binary vector set, where each vector has a one in an entry with a probability that is similar to the probabilities produced by the QS. By looking at when dependency occurs, we may be able to improve the algorithm by being able to use less vectors in the Gaussian elimination, which means the interval we are sieving can be shortened.

3. Constant Weight Vectors

Since the vectors used in Gaussian elimination are sparse, it was originally thought that a good way to model theses vectors was to assign a constant weight to each vector, meaning each vector has l ones from some fixed l. For this, we devised a computer program that would produce random k dimensional vectors with weight three, and calculate how many vectors were needed before a dependency was found. The program used Gaussian elimination to determine this.

Using this model, we got a threshold function for the probability that shows that you need as many vectors as is 92% to 93% of the dimension, or number of columns. It was previously shown by the second author in this paper, that, roughly, if m is much less than $k(1-\frac{e^{-l}}{log 2})$, then as $k \to \infty$, the probability taht m vectors are linearly dependent tends to zero.[1]

Figure 1 is an example of a weight three binary vector probability graph for dimension five hundred. Only the top eighty-five to ninety-five percentiles are pictured. As you can see, the lower bound for a high probability for dependency occurs when there are about 465 vectors, which is about 92%.

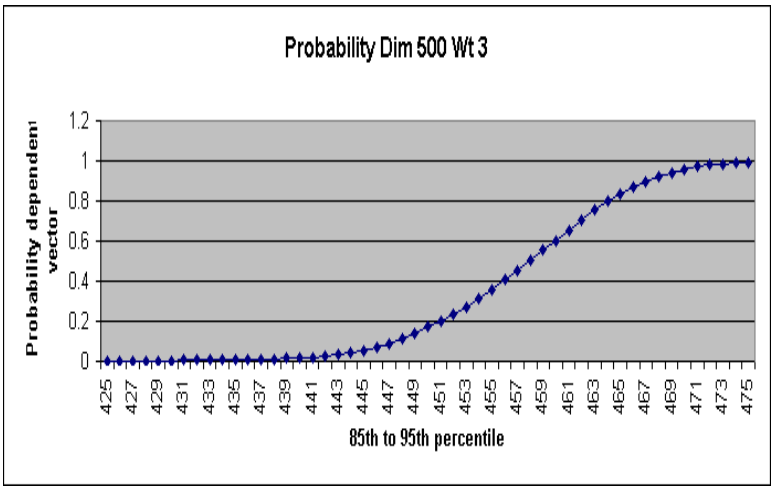

FIGURE 1. Probability Distribution for Weight 3 Dim 500 Vectors

4. Independent Weight Vectors

The constant weight binary vector model, in essence, says that it is as likely for there to be a one in the first column as it is the hundredth column. When relating back to the quadratic sieve, this model is saying that it is as equally likely for a number to be divisible by an odd power of two as it would be by an odd power of forty-three. Since this seems not to be true, this model should be altered so that it is closer to modeling what happens in reality.

The way that we decided to do this would be to have each entry have a different probability of having a one. So, the probability of getting an odd power of two is different from the probability of getting and odd power of three and so forth. The probability for each is as follows: Let p_i represent the prime associated with the i^{th} column of the matrix. Then the probability that the i^{th} column has a one in it is the probability that $p - i$ divides some number n to the first power minus the probability that p_i divides some number n to the second power (even power) and so forth and is shown below.

$$
\begin{aligned}
&= \frac{1}{p_i} - \frac{1}{p_i^2} + \frac{1}{p_i^3} - \frac{1}{p_i^4} + \dots \\
&= (\frac{1}{p_i} + \frac{1}{p_i^3} + \frac{1}{p_i^5} + \dots) - (\frac{1}{p_i^2} + \frac{1}{p_i^4} + \frac{1}{p_i^6} + \dots) \\
&= (\frac{1}{p_i})(1 + \frac{1}{p_i^2} + \frac{1}{p_i^4} + \dots) - (\frac{1}{p_i^2})(1 + \frac{1}{p_i^2} + \frac{1}{p_i^4} + \dots) \\
&= (\frac{1}{p_i} - \frac{1}{p_i^2})(1 + \frac{1}{p_i^2} + \frac{1}{p_i^4} + \dots) \\
&= (\frac{p_i - 1}{p_i^2})(\frac{1}{1 - \frac{1}{p_i^2}}) \\
&= \frac{p_i - 1}{p_i^2 - 1} = \frac{1}{p_i + 1}
\end{aligned}
$$

Given this, we edited our computer algorithm from the constant weight vector experiment to randomly generate these vectors with independent probabilities. This model gave us a dependency much earlier than before. In fact, for a dimension up through 25000 we got a dependency with less than 100 vectors. This is very different from the 92% we were getting earlier.

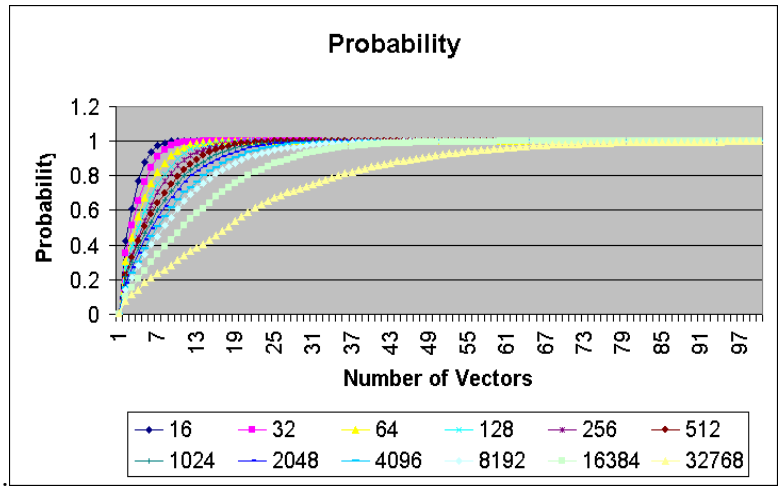

FIGURE 2. Probability of Dependency for first 100 vectors

A good upper bound for this model would be to find an upper bound for when a trivial dependency is found. A *trivial dependency* occurs when we have more vectors than the number of columns with a one in

some row. Our data shows that nontrivial dependencies occur before trivial dependencies. Therefore, having an upper bound for dependency occurring related to the trivial dependencies is appropriate. We know that since the probability that an entry in a vector has a one is $\frac{1}{p_i+1}$, we can deduce that the probability of getting a zero is $1 - \frac{1}{p_i+1}$. Then the expected number of columns with no ones, where j is the number of rows, is:

$$\sum_{i=1}^{k} \left(1 - \frac{1}{p_i + 1}\right)^j \simeq \sum_{i=1}^{k} 1 - e^{\frac{-j}{p_i+1}}$$

Thus, we want to find a function, $f(k)$, where k is the dimension of the vectors and $f(k)$ is the number of vectors in the vector space such that:

$$f(k) + \sum_{i=1}^{k} e^{\frac{-f(k)}{p_i+1}} > k$$

Due to the data we collected from our experiment, we believe this upper bound to be somewhere around $\sqrt{n}$ for n large enough, but this is what we're trying to prove....

References

[1] Calkin, Neil. "Dependent Sets of Constant Weight Binary Vectors." *Combin. Probab. Comput.* 6, no.3, 263-271 (1997).

[2] Landquist, Eric. "The Quadratic Sieve Factoring Algorithm." ¡http://www.math.uiuc.edu/ landquis/quadsieve.pdf¿, Dec 14, 2001 (verified July 8, 2004).

[3] Pomerance, Carl. "The Tale of Two Sieves." *Notices Amer. Math. Soc.* 43, no. 12, 1473-1485 (1996).